Modèles Graphiques Probabilistes pour l'Estimation de Densité en grande dimension : applications du principe Perturb & Combine pour les mélanges d'arbres

Sourour Ammar

▶ **To cite this version:**

Sourour Ammar. Modèles Graphiques Probabilistes pour l'Estimation de Densité en grande dimension : applications du principe Perturb & Combine pour les mélanges d'arbres. Informatique [cs]. Université de Nantes, 2010. Français. NNT : . tel-00568136

HAL Id: tel-00568136

https://theses.hal.science/tel-00568136

Submitted on 22 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE DE NANTES

ÉCOLE DOCTORALE

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE MATHEMATIQUES »

Année : 2010

Thèse de Doctorat de l'Université de Nantes

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

Sourour AMMAR KESSENTINI

Le10 décembre 2010

à Polytech'Nantes

Modèles Graphiques Probabilistes pour l'Estimation de Densité en grande dimension : applications du principe Perturb & Combine pour les mélanges d'arbres

Jury

Rapporteurs :	Salem BENFERHAT	Professeur à l'université d'Artois
	Nahla BEN AMOR	Maître de conférences habilitée à l' Institut Supérieur de Gestion de Tunis
Examinateurs :	Louis WEHENKEL	Professeur à l'université de Liège
	Stephane CANU	Professeur à l'INSA de Rouen
	Marc GELGON	Professeur à l'université de Nantes

Directeur de Thèse : Philippe LERAY
Laboratoire : Laboratoire d'Informatique de Nantes Atlantique (LINA)
Composante de rattachement du directeur de thèse : Polytech'Nantes

N° ED 503-117

Table des matières

Table des figures

Introduction générale

L'estimation de densité en grande dimension

La théorie des probabilités est un formalisme puissant et général qui a été appliquée avec succès dans divers domaines scientifiques et techniques. Dans le domaine du *machine learning*, les approches probabilistes ont prouvé qu'elles sont particulièrement fructueuses.

La tâche de l'estimation de densité revient à produire, à partir d'un ensemble de données d'observations, un modèle ou une description des ces données. Sous l'hypothèse que ces données ont été générées par un certain processus, le but de l'estimation de densité est alors de construire un modèle décrivant ce processus. Ce modèle estimé servira par la suite à prévoir les données qui vont être générées par ce processus.

Dans les applications actuelles, la complexité des systèmes réels ne cesse d'augmenter. Le nombre de variables décrivant ces systèmes est alors de plus en plus important ce qui rend difficile l'estimation de densité. En effet, le nombre de paramètres nécessaires pour une estimation (paramétrique) de la distribution ayant généré l'ensemble de données d'observations croit exponentiellement par rapport à la dimension du problème. De plus, l'augmentation de la taille de l'espace des paramètres conduit à une augmentation exponentielle de la complexité des calculs pour trouver un ensemble de paramètres optimaux. Les modèles graphiques probabilistes, dont nous parlerons dans le paragraphe suivant, fournissent une aide non négligeable pour lutter contre le problème de l'augmentation du nombre de paramètres en fournissant une factorisation de la loi jointe.

Le problème de grande dimension s'accentue du fait que le nombre d'observations avec lequel on effectue l'estimation de densité n'augmente pas dans les mêmes proportions, et reste même extrêmement faible dans certains domaines d'applications. Le principe du *Perturb et Combine*, initialement appliqué en classification,

permet de lutter contre ce genre de problèmes.

Les modèles graphiques probabilistes

Les modèles graphiques probabilistes profitent de leur pouvoir de modéliser graphiquement des dépendances et indépendances (conditionnelles) entre des sous-ensembles de variables ce qui permet de réduire le nombre de paramètres nécessaires pour apprendre une distribution de probabilité à partir d'un ensemble de données.

L'apprentissage de structure de modèles probabilistes revient essentiellement à trouver une factorisation maximale de la densité jointe d'un ensemble de variables aléatoires, ceci en se basant sur un ensemble d'observations jointes de ces variables [CDLS99]. Malheureusement, les algorithmes d'apprentissage de structure de modèles graphiques disponibles actuellement sont souvent restrictifs du point de vue de la nature de la structure qu'il cherchent, et ils ont une complexité algorithmique importante pour être applicables efficacement pour un très grand nombre de variables. De plus, le comportement de ces méthodes dans le cas d'un nombre réduit d'individus n'est pas connu. Il est possible que ces méthodes souffrent du phénomène de sur-apprentissage lorsque le nombre de variables est très grand par rapport au nombre d'individus, ce qui est le cas des problèmes actuels. Le principe appelé *Perturb et Combine* propose des techniques permettant de lutter contre ce problème de passage à l'échelle et celui de données de taille faible devant la dimension.

Principe du Perturb & Combine

Dans le contexte de l'apprentissage supervisé, un principe générique appelé *Perturb et Combine* a mené à beaucoup d'innovations fructueuses. Son idée principale est d'une part de perturber selon différentes directions un algorithme d'optimisation utilisé pour produire un prédicteur faible à partir d'un ensemble de données, et d'autre part combiner de façon appropriée un ensemble de prédicteurs (sous la forme d'un mélange de modèles) obtenus par plusieurs itérations de l'algorithme perturbé sur l'ensemble de données. La perturbation peut se faire selon deux niveaux : perturbation directe de l'algorithme d'apprentissage ou perturbation de l'ensemble de données. Dans ce principe, des ensembles de modèles aléatoires de performance faible ont été étudiés et utilisés avec succès pendant les

deux dernières décennies. Parmi les avantages de ces méthodes, citons l'amélioration potentielle du passage à l'échelle de leur algorithmes d'apprentissage (e.g. [GEW06]). Dans le contexte de l'estimation de densité, le Bagging (et le Boosting) de distributions normales a été proposé par [Rid02].

Organisation du document

Ce rapport est organisé autour de cinq chapitres. Dans le chapitre 1, nous présentons la notion générale de **l'estimation de densité** ainsi que les deux types d'approches permettant de faire l'estimation de densité : les méthodes paramétriques et les méthodes non-paramétriques. Enfin, nous discutons les différents problèmes rencontrés dans les espaces de grande dimension.

Le chapitre 2 nous présente **les modèles graphiques probabilistes** de différents points de vue. Nous commençons ce chapitre par introduire le principe général de ces modèles graphiques en mettant l'accent principalement sur la factorisation de la loi jointe ainsi que sur l'apport de ces modèles dans le cadre de l'estimation de densité. Nous étudions ensuite le cas particulier de modèles dont la structure du graphe est simple.

Ces modèles simples seront notre outil pour construire des mélanges d'arbres. Dans le chapitre 3, nous introduisons dans un premier temps le principe général des **mélanges de modèles** ainsi que le cadre général de cette approche *le Bayesian Model Averaging* où la taille du mélange est infinie. Nous présentons également les approches de mélanges de taille finie dont l'apprentissage à partir de données d'observation est basé sur l'algorithme EM. Ensuite, nous présentons un autre type d'approches basées sur le principe du *Perturb et Combine* pour l'apprentissage de mélanges de modèles finis. Enfin, nous discutons les différents approches présentées dans le cadre spécifique où les données sont de taille très faible devant la grande dimension du problème.

Après avoir présenté et discuté ces trois notions, nous décrivons nos contributions dans le cadre de l'estimation de densité par les modèles graphiques probabilistes en utilisant des mélanges de modèles simples selon le principe du *Perturb et Combine*. Dans le chapitre 4, nous considérons en premier temps **des mélanges de modèles simples** (arbres et polyarbres) générés aléatoirement selon une loi uniforme dans l'espace des modèles possibles. L'apprentissage du mélange ne se fait pas par l'algorithme EM classiquement utilisé pour apprendre la totalité des paramètres du mélange, mais selon un algorithme que nous proposons pour

apprendre les paramètres de chaque modèle simple indépendamment des autres. Ensuite, nous proposons un autre type de mélange où la perturbation porte sur les données d'apprentissage en utilisant le principe du Bagging, et les modèles simples sont appris à partir de données au moyen d'un algorithme classique d'apprentissage d'arbres optimaux. La deuxième partie de ce chapitre est consacrée aux résultats obtenus pour ces deux approches. Nous verrons que nos mélanges de modèles simples représentent une solution potentiellement intéressante pour l'estimation de densité, étant donné que nos mélanges passent à l'échelle et permettent d'obtenir de bons résultats par rapport aux méthodes de l'état de l'art. L'apprentissage de nos modèles de mélanges à base de Bagging, dont les résultats sont meilleurs, sont de complexité quadratique en fonction de la taille du domaine.

Ces différents résultats ont fait l'objet de plusieurs publications scientifiques : une première publication [ALDW08b] dans *The Annual Belgian-Dutch Machine Learning Conference*, une deuxième publication [ALW08] dans les *4èmes Journées Francophones sur les Réseaux Bayésiens*, une troisième publication [ALDW08a] lors du 4ème workshop européen *The European Workshop on Probabilistic Graphical Models* et une autre publication [ALDW09b] dans la 10ème conférence européenne *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* et aussi présentée [ALDW09a] lors de la *Conférence francophone sur l'APprentissage artificiel*.

Le chapitre 5 est alors consacré à nos dernières contributions, toujours à base du principe *Perturb et Combine*, avec la proposition d'**heuristiques sous-quadratiques de mélanges d'arbres** en appliquant une perturbation sur la procédure quadratique de recherche de l'arbre optimal. Ce deuxième type d'approches est ensuite testé et comparé à nos premiers mélanges ainsi que d'autres travaux concurrents. Nous verrons que nos heuristiques sous-quadratiques permettent de diminuer la complexité de construction de nos mélanges et représentent une alternative moins coûteuse que les précédents algorithmes.

Ce travail a fait le sujet d'autres publications scientifique : une première publication dans la *Conférence Internationale Computational Statistics* en 2010 [ALW10b], une deuxième publication [ALW10a] présentée lors des *5èmes Journées Francophones sur les Réseaux Bayésiens* et une troisième publication [ALSW10] dans le *Workshop on Probabilistic Graphical Models*.

Dans le chapitre 6, nous finissons par une conclusion générale pour ce travail de thèse ainsi que quelques perspectives.

Chapitre 1

L'estimation de densité

1.1 Introduction

Dans le domaine des probabilités et statistiques, l'estimation de la densité est la construction (recherche) d'une fonction qui approche une fonction de densité de probabilité, pour laquelle un ensemble de données a été observé. Nous introduisons dans ce premier chapitre l'estimation de densité en général ainsi que les différentes techniques et méthodes qui existent. Nous commençons dans la section 1.2 par présenter le principe général de l'estimation de densité ainsi que l'intérêt de cette notion. La section 1.3 est consacrée à la description des méthodes paramétriques et la section 1.4 décrit les méthodes d'estimation de densité non-paramétriques ainsi que leurs applications pour des exemples de lois. Nous présentons ensuite le contexte qui nous intéresse dans ce travail où le nombre d'observations est très petit devant la dimension du problème. Nous verrons dans la section 1.5 que la grande dimension des données pose des problèmes spécifiques en apprentissage statistique. Le problème de la dimension est généralement appelé « fléau de la dimension ». Nous verrons aussi dans cette même section les autres problèmes théoriques et pratiques causés par ce fléau de la dimension.

1.2 Principe général

Considérons une quantité aléatoire continue X qui possède une fonction de densité de probabilité $f(X)$. La spécification de la fonction $f(X)$ donne une description naturelle de la distribution de X, et permet de retrouver les probabilités

associées à X par l'équation suivante :

$$P(a < X < b) = \int_a^b f(x)dx \qquad pour\ tout\ a < b$$

Supposons maintenant que nous disposons d'un ensemble de données d'observations issu d'une fonction de densité de probabilité inconnue. L'estimation de densité, que nous considérons dans notre travail, est la construction, à partir des données observées, d'une fonction estimée qui représente (approche) cette fonction de densité.

Le problème évoqué dans notre travail est alors ***"Comment estimer une densité à partir d'un ensemble de données ?"***

Les approches d'estimation de densité se divisent en deux catégories : les approches *paramétriques* décrites dans la section 1.3 et les approches *non-paramétriques* décrites dans la section 1.4.

Intérêt de l'estimation de densité L'estimation de la densité est utilisée dans divers domaines d'applications, nous citons d'abord la simulation de nouvelles instanciations des variables aléatoires pour obtenir de nouvelles observations. Ensuite, dans le domaine de la reconnaissance de formes, l'estimation de la densité est utilisée pour faire de la classification par les modèles génératifs. Elle est utilisée aussi en optimisation avec par exemple les EDA (Estimation of Distribution Algorithms).

Nous commençons par définir un ensemble de notations que nous utiliserons dans la suite de ce chapitre. Soit $X_1, ..., X_n$ n observations équipondérées d'une variable aléatoire (réelle continue) X de densité f inconnue. Nous considèrerons dans tout le reste de ce rapport que les échantillons sont *iid* (indépendant et identiquement distribués). Le symbole $\widehat{f}$ sera utilisé pour noter l'estimateur de densité considéré.

1.3 Les approches paramétriques

1.3.1 Principe général

Dans le cas des modèles paramétriques, nous supposons que les données sont issues d'une famille de distributions paramétriques connue. L'avantage de l'approche paramétrique est que le modèle à estimer est défini par le moyen d'un

ensemble de paramètres notés θ (par exemple moyenne et variance pour la loi normale). Dans ces approches, la fonction densité dépend donc de la valeur de ces paramètres. Cette densité en un point x est couramment notée $f(x|\theta)$ ou $f_\theta(x)$.

Différentes méthodes existent pour l'estimation des paramètres décrivant les lois paramétriques, notamment la méthode du maximum de vraisemblance et la méthode d'estimation bayésienne. Il existe aussi la méthode des moments (en statistique), qui consiste à estimer les paramètres recherchés en égalisant certains moments théoriques (qui dépendent de ces paramètres) avec leurs contreparties empiriques.

Avant de passer à présenter ces approches d'estimation de densité paramétriques, nous introduisons certaines notions qui seront utiles par la suite.

Propriétés d'un estimateur

Un estimateur est considéré un "bon" estimateur s'il possède certaines "bonnes" propriétés [Alp04]. Soit X une variable aléatoire de densité de probabilité $f(x|\theta)$ et D un ensemble d'échantillons générés par cette densité. Soit $\widehat{\theta}$ un estimateur de θ. Un critère pour évaluer la qualité d'un estimateur $\widehat{\theta}$ est de calculer son *erreur quadratique moyenne* $r(\widehat{\theta}, \theta)$ donnée par :

$$r(\widehat{\theta}, \theta) = E\left[(\widehat{\theta} - \theta)^2\right]$$

Cette erreur quadratique moyenne s'exprime en fonction de la variance et du biais d'un estimateur :

$$r(\widehat{\theta}, \theta) = Var(\widehat{\theta}) + biais(\widehat{\theta})^2$$

Pour avoir un bon estimateur, il faut minimiser cette erreur donc minimiser sa variance et son biais.

Estimateur convergent Un estimateur est convergent s'il produit des estimations $\widehat{\theta}$ de plus en plus proches de la vraie valeur θ quand on considère des tailles d'échantillons de plus en plus grandes. Autrement dit, un estimateur est convergent en probabilité s'il tend en probabilité vers θ lorsque le nombre d'échantillons augmente infiniment.

Biais Dans la réalité, la taille des échantillons disponibles est limitée. Il est donc naturel de se demander quelle qualité est attendue d'un estimateur limité à des échantillons de taille donnée N.

Le biais d'un estimateur représente la quantité qui sépare la région centrale de sa distribution de la valeur vraie θ du paramètre à estimer. Cette quantité est donnée par l'équation suivante :

$$biais(\widehat{\theta}) = (E\left[\widehat{\theta}\right] - \theta)^2$$

La figure 1.1 est un exemple en une dimension, d'un estimateur $\widehat{\theta}$ d'un vrai paramètre θ à partir un ensemble d'échantillons d_i. Cette figure montre le biais de cet estimateur $\widehat{\theta}$ et sa variance.

Un estimateur dont la moyenne de la distribution (l'espérance) est égale à θ pour toute valeur de N est dit sans biais ou non-biaisé. Cette propriété se traduit par

$$E[\widehat{\theta}]_N = \theta \quad pour\ tout\ N$$

Pour une valeur de N donnée, un estimateur dont l'espérance n'est pas égale à θ est dit biaisé.

Variance la variance d'un estimateur mesure en moyenne comment les diverses estimations sont distribuées autour de leur valeur moyenne. La variance est données par l'équation suivante :

$$Var(\widehat{\theta}) = E\left[(\widehat{\theta} - E(\widehat{\theta})^2)\right]$$

La figure 1.1 montre la variance d'un estimateur $\widehat{\theta}$ pour un ensemble d'estimations.

Estimateur efficace Un paramètre peut avoir plusieurs estimateurs sans biais, dans ce cas il faudra en choisir un. L'estimateur le plus efficace sera celui ayant la plus faible variance.

Un estimateur est dit efficace s'il est sans biais. Il vérifie l'ensemble des conditions de Cramer-Rao [Cra99] et sa variance est égale à la borne de Cramer-Rao.

Si T est un estimateur asymptotiquement sans biais, et si $\lim_{N\to\infty} E(T) = \theta$ et $\lim_{N\to\infty} Var(T) = 0$ alors T est convergent.

1.3.2 La méthode du maximum de vraisemblance (MV)

Les méthodes d'estimation du maximum de vraisemblance sont des estimations ponctuelles puisqu'elles cherchent à trouver une valeur estimée $\widehat{\theta}$ pour un paramètre θ à partir d'un ensemble d'échantillons donné.

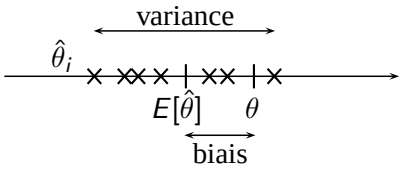


Figure 1.1. Biais et variance d'un estimateur

Elles sont attractives pour plusieurs raisons. D'abord, elles possèdent de bonnes propriétés de convergence et d'efficacité quand le nombre d'échantillons est très grand. Les estimateurs du MV ont asymptotiquement la variance la plus faible parmi tous les estimateurs sans biais. De plus, l'estimation par maximum de vraisemblance est plus simple que les méthodes alternatives, telles que les méthodes bayésiennes. En effet, du point de vue complexité, les méthodes de maximum de vraisemblance ne nécessitent que des techniques de calculs différentiels ou une recherche de gradient, tandis que les méthodes bayésiennes peuvent nécessiter des intégrations multidimensionnelles complexes.

Une caractéristique majeure indésirable du MV est que ses propriétés avec des échantillons finis peuvent être très différentes de ses propriétés asymptotiques. Bien qu'elles soient convergentes, les estimations des paramètres par MV sont typiquement biaisées.

Soit X une variable aléatoire de densité de probabilité $f(x|\theta)$. Soit $D = \{x_1, x_2, \ldots, x_n\}$ un échantillon de données *iid* généré par la densité de probabilité $f(x|\theta)$ avec les paramètres $\theta \in \Omega$. Dans le cas où les variables aléatoires sont indépendantes, la fonction de vraisemblance associée L (vraisemblance de θ par rapport à D) est définie sur Ω par le produit de densités de probabilité individuelles (Equation 1.1)

$$L(\theta) = f(D|\theta) = \prod_{j=1}^{n} f(x_j|\theta) \tag{1.1}$$

La méthode du maximum de vraisemblance consiste à trouver les valeurs $\widehat{\theta}$ de θ qui maximisent la vraisemblance $L(\theta)$ d'avoir obtenu l'échantillon utilisé : $\widehat{\theta} = \max_{\theta \in \Omega} L(\theta)$. Cette estimation correspond à la valeur de θ qui supporte le mieux l'ensemble de données observées.

Grâce à la monotonie de la fonction *Logarithme*, $\widehat{\theta}$ peut être trouvé en maximisant le logarithme de la fonction de vraisemblance $l(\theta)$ (appelé fonction *log-vraisemblance*). Cela possède l'avantage en calcul de remplacer un produit par

une somme.

$$l(\theta) = log(L(\theta)) = \sum_{j=1}^{n} log(f(x_j|\theta)) \tag{1.2}$$

On peut écrire la solution sous la forme :

$$\widehat{\theta}_{MV} = arg \max_{\theta \in \Omega} \ l(\theta)$$

Si le nombre de paramètres de θ est égal à p, alors le vecteur de p *composantes* de θ est : $\theta = (\theta_1, \ldots, \theta_p)^t$, et ∇_θ est l'opérateur gradient.

$$\nabla_\theta = \begin{bmatrix} \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_p} \end{bmatrix}$$

D'après l'équation 1.2, on a :

$$\nabla_\theta l = \sum_{j=1}^{n} \nabla_\theta log(f(x_j|\theta))$$

La solution $\widehat{\theta}$ doit satisfaire la condition :

$$\nabla_\theta l = 0 \tag{1.3}$$

Il est à noter qu'une solution $\widehat{\theta}$ trouvée peut ne pas représenter un vrai maximum global. Elle peut représenter un maximum ou un minimum local ou encore un point d'inflexion de la fonction $l(\theta)$. Une vérification de chaque solution trouvée est nécessaire pour s'assurer qu'il s'agit d'un vrai maximum, ou un calcul des dérivées secondes peut confirmer la nature de l'optimum trouvé.

L'estimation de maximum de vraisemblance a l'avantage de simplicité et rapidité de son calcul si l'équation 1.3 est simple à résoudre comme par exemple dans le cas des familles exponentielles. Le principe du maximum de vraisemblance fourni une approche d'estimation bien connue dans le cas de distributions normales et plusieurs autres problèmes. Toutefois, dans le cas de problèmes complexes, un estimateur de maximum de vraisemblance peut devenir inapproprié ou peut même ne pas exister. En effet, dans le cas où l'équation 1.3 ne peut pas être analytiquement résolue, il faut avoir recours à des méthodes itératives comme la méthode de *Newton-Raphson*, la *méthode des scores* ou encore l'algorithme *Expectation-Maximization* pour essayer de trouver un maximum de la fonction de vraisem-

blance, ce qui n'est pas toujours faisable.

De point de vue interprétabilité, le maximum de vraisemblance est facile à comprendre et à interpréter du moment où cette méthode retourne un seul modèle optimal pour l'ensemble d'échantillons disponibles.

1.3.3 L'estimation Bayésienne : maximum a posteriori (MAP)

Dans certains cas, un expert du domaine peut avoir des connaissances (informations) *a priori* sur certaines variables ou même sur le problème en entier. Ces informations sont très utiles et doivent être exploitées et introduites en tant qu'*a priori* sur les paramètres θ, surtout dans le cas où il n'y a que très peu de données. Les informations a priori n'indiquent pas exactement les valeurs des paramètres (dans le cas contraire, les données seraient inutiles), l'incertain sera alors modélisé en considérant θ comme variable aléatoire et en définissant une densité a priori $f(\theta)$ pour cette variable.

La *densité a priori*, $f(\theta)$, nous indique la valeur que doit avoir θ avant de voir les données. On combine ensuite cette information *a priori* avec les informations issues des données, notamment la densité obtenue par maximum de vraisemblance $f(D|\theta)$, en utilisant la règle de Bayes. On obtient ainsi une densité a posteriori de θ.

$$f(\theta|D) = \frac{f(D|\theta)f(\theta)}{f(D)} = \frac{f(D|\theta)f(\theta)}{\int_{\theta' \in \Omega} f(D|\theta')f(\theta')d\theta'} \tag{1.4}$$

L'estimation de la densité en un point x est donnée par l'équation suivante :

$$f(x|D) = \int_{\Omega} f(x|\theta)f(\theta|D)d\theta \tag{1.5}$$

L'estimation bayésienne est alors une moyenne (espérance) des prédictions en utilisant toutes les valeurs possibles de θ, pondérées par leur probabilité. Le calcul de cette intégrale de l'équation 1.5 peut être un problème difficile à résoudre. Dans le cas où ce calcul n'est pas faisable, on peut approcher l'intégrale par un seul point. L'estimation du maximum a posteriori (MAP) peut rendre le calcul plus facile en faisant l'hypothèse que $f(\theta|D)$ présente un pic.

$$\widehat{\theta}_{MAP} = arg\max_{\theta} \ f(\theta|D)$$

Ainsi, la densité dans l'équation 1.5 est remplacée par un seul point comme

suit :

$$f(x|D) \simeq f(x|\widehat{\theta}_{MAP})$$

Une autre possibilité pour résoudre le problème de l'équation 1.5 est de considérer l'estimateur Bayésien appelé *espérance a posteriori (EAP)*. Cet estimateur consiste à calculer l'espérance (moyenne) de toutes les valeurs possibles du paramètre à estimer. L'estimateur EAP est alors donné par la formule suivante :

$$\widehat{\theta}_{EAP} = E_\theta \left[f(\theta|D) \right] = \int \theta f(\theta|D) d\theta$$

Le choix de la loi a priori est une des difficultés majeures de l'approche bayésienne. Il est généralement guidé par les connaissances a priori que l'on peut avoir du domaine étudié. Lorsque la connaissance a priori est très précise elle peut impliquer l'utilisation d'une loi a priori entièrement spécifiée. En l'absence d'information a priori, on peut également utiliser une loi a priori non informative tel que une loi uniforme.

Les calculs qui mènent à la loi a posteriori, donc à l'estimation bayésienne des paramètres θ ne sont pas toujours simples et la difficulté provient du calcul des intégrales de l'équation 1.4. Un choix judicieux de la densité a priori permet de résoudre analytiquement certaines de ces intégrales (ou même toutes). Lorsque le calcul analytique est impossible, on doit utiliser des méthodes d'approximation, soit basées sur des approximations numériques de l'intégrale à calculer, soit basées sur la simulation selon les lois d'intérêt (loi a posteriori ou loi prédictive a posteriori).

Il est possible d'avoir recours à la famille des lois a priori conjuguées dont l'intérêt est d'assurer le calcul analytique de la densité a posteriori. En effet, elles sont définies de telle sorte que la loi a posteriori appartient à la même famille de lois que la loi a priori : si la loi a priori est de densité $f_\gamma(\theta)$, alors la loi a posteriori sera de densité $f_{\gamma'}(\theta)$ où γ' s'exprime aisément en fonction de γ et des données. Les lois a priori conjuguées sont connues pour un certain nombre de modèles classiques, par exemple, la distribution a priori conjuguée à une distribution gaussienne (univariée) est une gaussienne et celle conjuguée à une distribution multinomiale (multivariée) est une distribution de Dirichlet. Ce concept des distributions a priori conjuguées a été introduit par [RS61].

Cette estimation bayésienne est mieux adaptée dans le cas où des connaissances a priori sur les variables sont fournies et dans le cas où on ne dispose que de très peu de données.

1.3.4 Exemples de modèles uni-variés

1.3.4.1 Variable discrète : loi de Bernoulli

Forme de la loi

Soit une variable aléatoire X discrète qui suit une loi de Bernoulli de paramètre p notée $B(p)$. Dans une loi de Bernoulli, il existe deux sorties possibles : un évènement se produit ou non. Si l'évènement se produit, la variable aléatoire de Bernoulli X prend la valeur 1 avec une probabilité p, l'évènement ne se produit alors pas avec une probabilité $1-p$ et la variable aléatoire X prend la valeur 0. Cela est défini par :

$$P(x) = p^x(1-p)^{1-x}, x \in \{0,1\}$$

p est le seul paramètre à estimer pour définir une loi Binomiale.

Estimation par Maximum de vraisemblance

Soit un échantillon iid $D = \{x_t\}_{t=1}^N$ où $x_t \in \{0,1\}$, pour calculer l'estimateur de Maximum de vraisemblance $\hat{p}$, nous définissons la fonction *log-vraisemblance*

$$\begin{aligned} l(p) &= \log \prod_{t=1}^{N} p^{x_t}(1-p)^{1-x_t} \\ &= \sum_t x_t \log p + \left(N - \sum_t x_t\right) \log(1-p) \end{aligned}$$

$\hat{p}$ qui maximise le *log-vraisemblance* peut être calculé en résolvant $dl/dp = 0$.

$$\hat{p} = \frac{\sum_t x_t}{N}$$

L'estimateur de p est donné par le nombre d'occurrences de l'évènement divisé par le nombre total d'individus.

Estimation par MAP

La distribution a priori conjuguée pour une loi de Bernoulli est une distribution *Beta* de paramètres α et β, notée $Beta(\alpha, \beta)$.

Soit un échantillon iid $D = \{x_t\}_{t=1}^N$ où $x_t \in \{0,1\}$, et une distribution a priori de type Beta $Beta(\alpha, \beta)$ définie par

$$Beta(p; \alpha, \beta) = \gamma p^{\alpha-1}(1-p)^{\beta-1}$$

avec $\gamma = \Gamma(\alpha+\beta)/\Gamma(\alpha)\Gamma(\beta)$ est une constante qui ne dépend pas de p.

Pour estimer le Maximum a posteriori de p, il faut maximiser la fonction

$$f(x|p)Beta(p;\alpha,\beta) = p^x(1-p)^{1-x}\gamma p^{\alpha-1}(1-p)^{\beta-1}$$

L'estimateur MAP $\hat{p}_{MAP}$ de p est donné par

$$\hat{p}_{MAP} = \frac{\sum_t x_t + \alpha - 1}{N + \alpha + \beta - 1}$$

1.3.4.2 Variable discrète : loi multinomiale

Une variable X discrète qui peut avoir plusieurs valeurs $\{1, 2, \ldots, r\}$ est dite multinomiale.

L'apprentissage des paramètres de sa fonction de probabilité à partir de données est similaire au cas des variables de Bernoulli. L'estimateur de MV est donnée par :

$$P(X = k) = \frac{N_k}{N}$$

où N_k et le nombre d'occurrences de x_k dans D.

La loi a priori conjuguée dans ce cas est une distribution de Dirichlet (voir définition 1) de paramètres $\{\alpha_1, \ldots, \alpha_r\}$.

Définition 1 (*Distribution de Dirichlet*)

Une distribution de Dirichlet d'ordre $K \geq 2$, avec les paramètres $\alpha_1, \ldots, \alpha_K > 0$. Elle est définie par

$$f(p_1, \ldots, p_{K-1}; \alpha_1, \ldots, \alpha_K) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} (p_i^{\alpha_i - 1})$$

pour tout $p_1, \ldots, p_{K-1} > 0$ vérifiant $p_1 + \ldots + p_{K-1} < 1$, où p_K est une abréviation pour $1 - p_1 - \ldots - p_{K-1}$ et $s = \sum \alpha_K$.

Avec $B(\alpha)$ est une fonction beta multinomiale.

L'estimateur MAP de la fonction de probabilité de X est donné par :

$$P(X = k|D) = \frac{N_k + \alpha_k - 1}{\sum_k (N_k + \alpha_k - 1)}$$

1.3.4.3 Variable continue : loi normale

Forme de la loi et paramètres

Soit X une variable aléatoire continue gaussienne qui suit une loi normale de moyenne μ et de variance σ^2, notée par $N(\mu, \sigma^2)$ et sa fonction densité est :

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], -\infty < x < \infty$$

Pour définir une estimation de la loi normale, il y a alors deux paramètres à estimer : la moyenne μ et la variance σ^2.

Estimation par Maximum de vraisemblance

Soit un échantillon iid $D = \{x_t\}_{t=1}^N$ où $x_t \sim N(\mu, \sigma^2)$. Le log de vraisemblance de cet échantillon gaussien est

$$l(\mu, \sigma) = -\frac{N}{2}\log(2\pi) - N\log\sigma - \frac{\sum_t (x_t - \mu)^2}{2\sigma^2}$$

L'estimateur Maximum de vraisemblance des paramètres de cette loi normale est donné par

$$\hat{\mu} = \frac{1}{N}\sum_{t=1}^{N} x_t$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{t=1}^{N}(x_t - \hat{\mu})^2$$

La figure 1.2 montre un exemple d'estimation de la densité d'une loi normale à partir d'un ensemble d'échantillons. La figure du haut montre l'ensemble des points en une dimension et des gaussiennes candidates. La figure du milieu montre la fonction de vraisemblance $L(\theta)$ en fonction de la moyenne. La figure du bas représente la fonction log-vraisemblance $l(\theta)$. Sur cette figure, la valeur $\widehat{\theta}$ qui maximise $L(\theta)$ maximise aussi $l(\theta)$.

Estimation par MAP

La distribution a priori conjuguée pour une loi normale est la distribution normale. Soit un échantillon iid $D = \{x_t\}_{t=1}^N$ où $x_t \sim N(\mu, \sigma_v^2)$ et une distribution gaussienne a priori de μ $N(0, \sigma_m^2)$. Pour estimer le Maximum a posteriori de μ

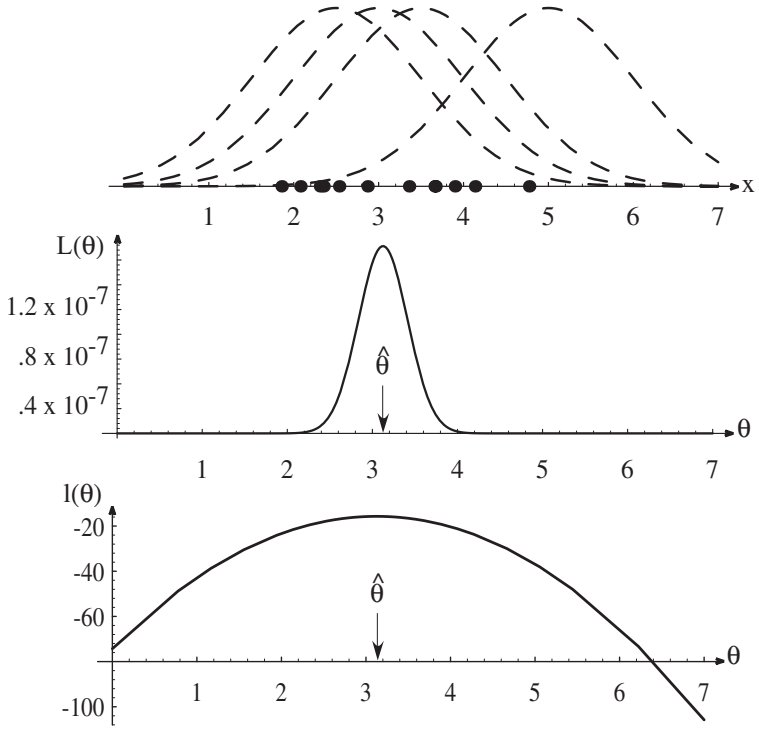


Figure 1.2. Figure issue de [DHS01] : Le graphe du haut montre plusieurs points d'apprentissage en une dimension, supposés générés par une loi gaussienne de variance particulière, mais de moyenne inconnue. Quatre parmi le nombre infini de distributions sources candidates sont représentées en trait discontinu. La figure du milieu présente la vraisemblance $L(\theta)$ comme une fonction de la moyenne. Si nous disposions d'un nombre infini de points, cette vraisemblance serait très étroite. La valeur qui maximise la vraisemblance est notée $\hat{\theta}$; elle maximise aussi le log-vraisemblance $l(\theta)$, représenté dans la figure du bas.

supposant que σ est connu, il faut maximiser la fonction

$$\begin{aligned} f(\mu)f(x|\mu) &= \pi(\mu)L(\mu) \\ &= \frac{1}{\sqrt{2\pi\sigma_m}} \exp\left(-\frac{1}{2}\left(\frac{\mu}{\sigma_m}\right)^2\right) \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi\sigma_v}} \exp\left(-\frac{1}{2}\left(\frac{x^j-\mu}{\sigma_v}\right)^2\right) \end{aligned}$$

Cela revient à maximiser μ dans l'équation suivante

$$\sum_{j=1}^{N}\left(\frac{x_j-\mu}{\sigma_v}\right)^2 + \left(\frac{\mu}{\sigma_m}\right)^2$$

L'estimation MAP de μ est alors donnée par

$$\hat{\mu}_{MAP} = \frac{\sigma_m^2}{N\sigma_m^2+\sigma_v^2}\sum_{j=1}^{N} x_j$$

1.3.5 Exemples de modèles multivariés usuels

Dans le cas où l'on dispose de plusieurs variables correspondant au problème traité, ce problème est dit *multivarié*. Soit $\mathbb{X} = (X_1, \ldots, X_n)$ un vecteur aléatoire. Si les éléments X_i sont des variables aléatoires univariés avec des fonctions de distributions $P_{X_i}(x_i)$, la distribution de $\mathbb{X}$ est dite *une distribution multivariée*. Si on dispose de n variables, l'ensemble de données D peut s'écrire sous la forme d'une matrice :

$$D = \begin{pmatrix} X_1^1 & X_2^1 & \ldots & X_n^1 \\ X_1^2 & X_2^2 & \ldots & X_n^2 \\ \vdots & & & \\ X_1^N & X_2^N & \ldots & X_n^N \end{pmatrix}$$

où les colonnes correspondent aux n variables et les lignes correspondent aux N individus.

1.3.5.1 Variables discrètes

Si les X_i sont discrets alors $\mathbb{X}$ est un vecteur aléatoire discret et sa fonction de probabilité est donnée par :

$$P_X(x) = P(X_1 = x_1, \ldots, X_n = x_n)$$

Pour estimer les paramètres de cette fonction de probabilité de $\mathbb{X}$, nous commençons par considérer un exemple de deux variables binaires ayant chacune une distribution de Bernoulli, donc chaque variable peut prendre deux valeurs possibles 0 et 1 : ainsi $\mathbb{X} = \{X_1, X_2\}$.

Les paramètres de cette distributions s'écrivent comme suit :

$$\theta_{ij} = P(X_i = x_i; X_j = x_j)$$

Pour estimer les paramètres de la distribution de $\mathbb{X}$, il faut estimer trois $(2^2 - 1)$ probabilités θ_{11}, θ_{21} et θ_{12}. avec $\theta_{22} = 1 - (\theta_{11} + \theta_{21} + \theta_{12})$.

L'estimateur MV de ces paramètres θ_{ij} est donnée par le nombre N_{ij} d'occurence de x_{ij} dans la base de données :

$$\hat{\theta}_{ij} = \frac{N_{ij}}{N}$$

Ce raisonnement reste valable pour n variables binaires avec $2^n - 1$ paramètres à estimer. L'estimation de ces paramètres devient alors très coûteuse avec l'augmentation du nombre de paramètres qui est exponentiel en fonction de n. Ce nombre peut encore augmenter si les variables sont multinomiales.

Dans ce cas, il est intéressant de recourir à des méthodes permettant de réduire le nombre total de paramètres nécessaire pour définir la distribution cible. Une solution intuitive est d'essayer d'utiliser des indépendances entre les variables afin de pouvoir diminuer le nombre total des paramètres à estimer, ce que les modèles graphiques permettent de faire.

1.3.5.2 Variable continues : loi gaussienne multivariée

Forme de la loi et paramètres

Soit $(x_1, \ldots, x_n)$ une séquence d'échantillons *iid* de variables aléatoires. Dans le cas où la distribution de données considérée est une distribution gaussienne (normale) multivariée notée $N(\mu, \Sigma)$, les paramètres à estimer sont $\theta = (\mu, \Sigma)$, où Σ est la matrice de variance-covariance (définie positive) de f et μ est son vecteur moyenne. Cette fonction de densité de probabilité s'écrit sous la forme :

$$f_\theta(x) = (2\pi)^{-n/2} \det(\Sigma)^{-1/2} \exp(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)) \tag{1.6}$$

avec $\det(\Sigma)$ est le déterminant de Σ, et $J = \Sigma^{-1}$ est l'inverse de la matrice de covariance, appelée *matrice d'information.*

Pour définir un estimateur d'une loi gaussienne multivariée, il faut estimer les deux paramètres : le vecteur moyenne μ et la matrice de variance-covariance Σ. L'estimation de ces derniers nécessite le calcul de plusieurs paramètres, certaines hypothèses peuvent être considérées pour diminuer ce nombre de paramètres et ensuite simplifier les calculs. Parmi ces hypothèses, il est possible de considérer que la matrice de variance-covariance est diagonale ou sphérique.

Estimation par Maximum de vraisemblance

Supposons en premier temps que seul le vecteur moyenne μ est inconnu. Dans ce cas, la vraisemblance de la fonction f est donnée par la formule suivante :

$$L(\mu, \Sigma) = \prod_{i=1}^{n} (2\pi)^{-n/2} \det(\Sigma)^{-1/2} \exp(-\frac{1}{2}(x_i - \mu)^T \Sigma^{-1}(x_i - \mu)) \tag{1.7}$$

Et la fonction log de vraisemblance de f est

$$\log(f(x_k|\mu)) = -\frac{1}{2} \log\left[(2\pi)^n |\Sigma|\right] - \frac{1}{2} (x_k - \mu)^t \Sigma^{-1}(x_k - \mu)$$

L'estimation par maximum de vraisemblance de μ est donnée par la moyenne des échantillons :

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Supposons maintenant que les deux paramètres μ et Σ sont inconnus.

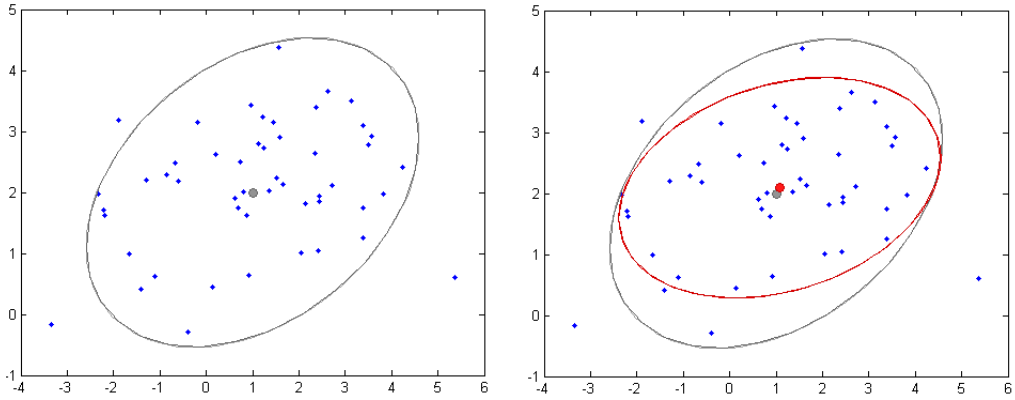
L'estimation par maximum de vraisemblance de μ est donnée par la moyenne des échantillons :

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

La valeur de Σ qui maximise le log de vraisemblance de f est alors donnée par :

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})(x_i - \hat{\mu})^T \tag{1.8}$$

Dans la figure 1.3, (a) représente une distribution gaussienne de deux dimensions dont la moyenne est $[1; 2]$ et la matrice de variance-covariance est $[4\ 1; 1\ 2]$, et un ensemble de 50 points générés selon cette distribution. (b) représente (en rouge) l'estimation maximum de vraisemblance correspondante de moyenne $[1.08; 2.09]$ et de matrice de variance-covariance $[3.73\ 0.537; 0.537\ 1.01]$, la moyenne de cette es-



(a) (b)

Figure 1.3. (a) une distribution gaussienne de dimension 2 et un ensemble de points générés par cette distribution, le point gris au milieu représente la moyenne de cette gaussienne. (b) l'estimation par maximum de vraisemblance à partir des donnée, cette estimation est une distribution gaussienne de dimension 2 dessinée en rouge, le point rouge au milieu est la moyenne de cette gaussienne.

timation est marquée en rouge.

Estimation par MAP

La distribution a priori conjuguée pour une loi gaussienne multivariée dont la moyenne et la variances sont inconnues est une distribution de Wishart normale. La fonction a priori est alors donnée par

$$P(\mu, \Sigma) = P(\mu)P(\Sigma|\mu) = N(\mu, \tau^{-1} I_d) \times W(\Sigma|\mu; \Lambda, \alpha)$$

avec $W(\Sigma|\mu; \Lambda, \alpha) \propto |\Sigma|^{(\alpha-d)/2} \exp(-\frac{1}{2} tr(\Lambda\Sigma))$

Les estimateurs MAP $\widehat{\mu}_{MAP}$ et $\widehat{\Sigma}_{MAP}$ sont données par [Mur03] :

$$\widehat{\mu}_{MAP} = \frac{\tau + \sum_{i=1}^{n} / n}{\tau + 1}$$

1.4 Les approches non-paramétriques

Dans le cas des méthodes paramétriques, on suppose que l'expression littérale de $f(x|D)$ est connue. Cette hypothèse forte, si erronée, peut conduire à de très mauvais résultats. Dans le cas des méthodes non paramétriques, les lois ne sont pas définies de manière claire, mais il existe parfois des paramètres à régler.

1.4.1 Principe général

Les méthodes non paramétriques prennent en compte les échantillons et leur répartition spatiale dans l'espace des paramètres afin de produire une estimation de $f(x|D)$ plus proche de la réalité. Elles estiment la densité de probabilité directement à partir des données sans se donner aucune hypothèse a priori sur la distribution des données. Il s'agit alors de calculer la probabilité en chaque point directement à partir des observations sans chercher à construire une fonction de distribution définie sur tout le domaine.

Nous présenterons dans la suite les méthodes non-paramétriques d'estimation de densité les plus connues. Nous nous inspirons dans cela du livre [Sil86]. Étant donné que toutes les méthodes multivariées sont des généralisations des méthodes univariées, nous présentons les différentes méthodes dans le cas univarié en montrant ensuite comment cette généralisation peut se faire.

1.4.2 Les histogrammes

Les histogrammes représentent la méthode d'estimation de densité la plus ancienne et la plus couramment utilisée.

Dans le cas discret, cette méthode revient à compter le nombre d'observations de l'échantillon qui sont égales à x. $\widehat{f}$ est donnée par la formule suivante :

$$\widehat{f}(x) = \frac{1}{n} \sum_{n}^{i} I(x_i = x)$$

où $I(\cdot)$ est la fonction indicatrice : elle vaut 1 lorsque son argument est vrai.

Dans la cas continu, on ne peut plus compter le nombre d'échantillons égal à x. Considérons une origine x_0 et un paramètre positif h. Considérons une boîte centrée en x_0, et de largeur h. Le nombre d'échantillons égal à x est alors donné par le nombre d'observations tombant dans cette boite. $\widehat{f}$ est alors donnée par la formule suivante :

$$\widehat{f}(x) = \frac{1}{nh} I(-\frac{1}{2} \leq \frac{x_i - x}{h} \leq \frac{1}{2})$$

Le choix de x_i et de h doit dépendre des échantillons. Le paramètre h contrôle le niveau de lissage de l'estimation et doit être recherché avec soin. Dans ce cas d'estimation, $\widehat{f}$ a une allure de constante par morceaux. Les histogrammes sont des estimateurs très faciles à utiliser mais peu réguliers. Les histogrammes peuvent êtres généralisés en variant le paramètre h.

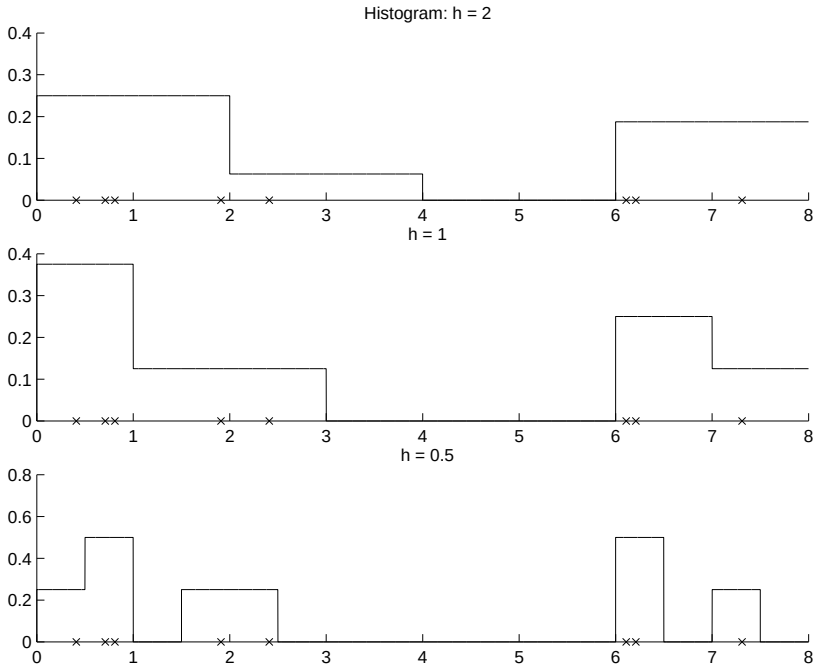


Figure 1.4. Histogrammes pour différentes valeurs de la largeur h.
Tiré de [Alp04].

1.4.3 L'estimateur naïf

L'estimateur naïf est initialement introduit par Fix et Hodges(1951) dans un rapport technique qui n'a jamais été publié par ses auteurs. [Ros56] est le premier article publié qui a évoqué cet estimateur naïf.

A partir de la définition d'une densité de probabilité, si la variable aléatoire X possède une densité f, alors

$$f(x) = lim_{h->0} \frac{1}{2h} P(x - h < X < x + h). \tag{1.9}$$

Pour n'importe quelle valeur de h, nous pouvons estimer $P(x-h < X < x+h)$ par la proportion d'échantillons dans l'intervalle $(x - h, x + h)$. Ainsi, un estimateur naturel $\widehat{f}$ de la densité est donné par le choix d'un petit nombre h. Cet estimateur est alors appelé l'estimateur naïf et est donné par l'équation suivante

$$\widehat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h}\ w\left(\frac{x - X_i}{h}\right).$$

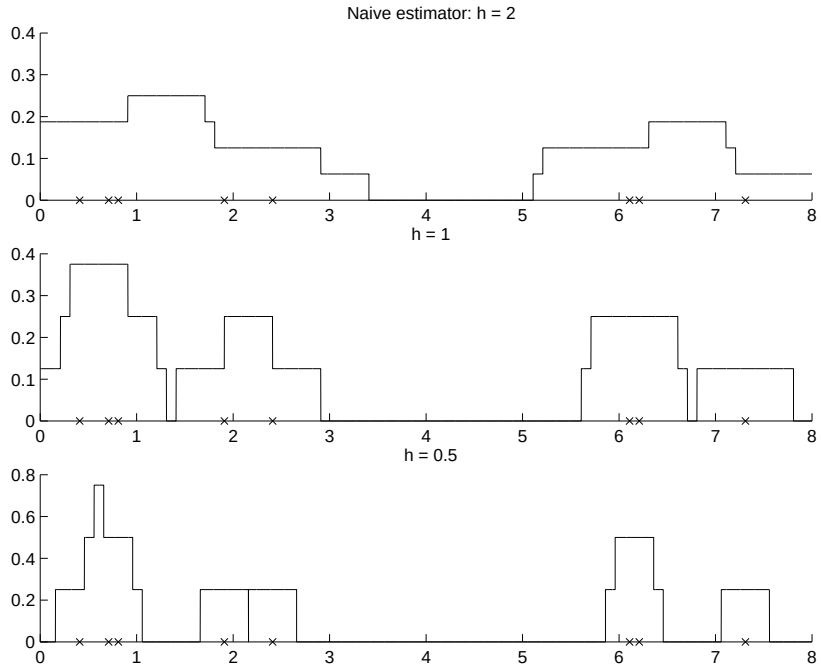


Figure 1.5. Estimateur naïf pour différentes valeurs de la largeur h.
Tiré de [AIp04].

Avec w, est une fonction de poids donnée par :

$$w(x) = \begin{cases} \frac{1}{2} & \text{si } |x| < 1 \\ 0 & \text{sinon.} \end{cases}$$

L'estimateur naïf peut être vu comme un histogramme où chaque point est le centre d'un intervalle d'échantillonnage. Ainsi cela revient à construire un histogramme à partir d'un choix particulier de la position des boîtes. Le choix de la largeur de chaque boîte reste géré par le paramètre h qui contrôle la qualité de lissage des données pour produire l'estimation.

L'estimateur naïf n'est pas satisfaisant de point de vue présentation. Il ressort à partir de la définition que $\widehat{f}$ n'est pas une fonction continue, mais présente des sauts aux points $X_i \pm h$ et une dérivée nulle ailleurs. De plus, le choix de la largeur h affecte significativement l'estimateur : Si la valeur de h est faible, l'estimation présentera de nombreux pics et si la valeur de h est grande, estimation sera plus douce mais moins précise.

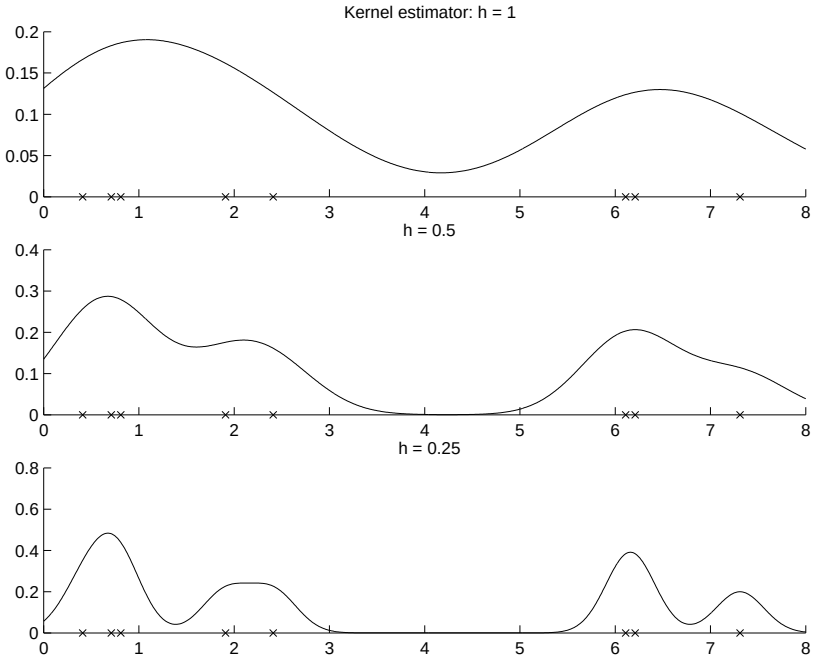


Figure 1.6. Estimation par noyau pour différentes valeurs de la largeur h.
Tiré de [Alp04].

1.4.4 La méthode à noyau

Il est facile de généraliser l'estimateur naïf pour résoudre certaines difficultés mentionnées dans la section précédente. Il s'agit de remplacer la fonction de poids w par une *fonction noyau* (*Kernel function* en anglais) K qui satisfait la condition

$$\int_{-\infty}^{\infty} K(x)dx = 1$$

Le plus souvent, K est choisie comme étant une fonction de densité de probabilité unimodale symétrique, par exemple une fonction gaussienne standard (espérance nulle et variance unitaire).

Par analogie avec la définition de l'estimateur naïf, l'estimation par noyau K [Ros56] est définie par

$$\widehat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right). \tag{1.10}$$

où h est la largeur de la fenêtre, appelé aussi *paramètre de lissage*.

L'estimation par la méthode du noyau est continue. En effet, la boîte centrée

en x et de largeur h (dans la méthode d'estimateur naïf) est remplacée par une courbe en cloche centrée en x. Plus une observation est proche du point de support x plus la courbe en cloche lui donnera une valeur numérique importante. A l'inverse, les observations trop éloignées de x se voient affecter une valeur numérique négligeable. L'estimateur est formé par la somme (ou plutôt la moyenne) des courbes en cloche.

1.4.5 La méthode des k plus proches voisins

La classe des estimateurs par les plus proches voisins (notée *k-ppv*), initialement introduite par [LQ65], représente une tentative d'adapter le degré de lissage de la densité 'locale' des données. Le degré de lissage est contrôlé par un entier k, choisi pour être considérablement plus petit que la taille de l'échantillon ; typiquement $k \approx n^{1/2}$. On définit la distance $d(x, y)$ entre deux points, usuellement elle vaut $|x - y|$, et pour tout t on définit

$$d_1(t) \leq d_2(t) \leq \ldots \leq d_n(t)$$

les distances, rangées dans un ordre croissant, du point t aux autres points de l'échantillon.

La $k^{ème}$ estimation de densité par la méthode du plus proche voisin est ainsi définie par :

$$\widehat{f}(t) = \frac{k}{2nd_k(t)} \tag{1.11}$$

Du moment où l'estimateur naïf est basé sue le nombre d'observations tombant dans la boite centrée au point d'intérêt et de largeur fixe, l'estimateur à base des plus proches voisins est inversement proportionnel à la taille de la boite utilisée pour contenir un certain nombre d'observations. Comme dans le cas de l'estimateur naïf, l'estimation par les plus proches voisins comme définie par l'équation 1.12 n'est pas une courbe lisse. La fonction $d_k(t)$ peut facilement être vue comme continue, mais sa dérivée sera discontinue. Contrairement à la méthode d'estimation par noyau, l'estimation par *k-ppv* ne sera pas une distribution de probabilité.

Il est possible de généraliser la méthode *k-ppv* pour produire une estimation en relation avec la méthode des noyaux. La méthode *k-ppv généralisée* est définie

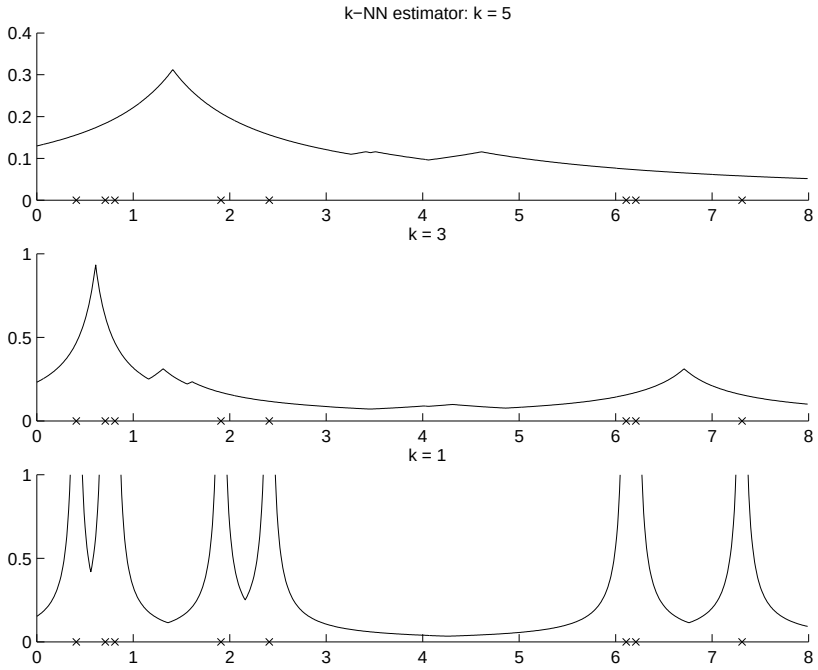


Figure 1.7. Estimation par k-ppv pour différentes valeurs de k.
Tiré de [Alp04].

par

$$\widehat{f}(t) = \frac{1}{nd_k(t)} \sum_{i=1}^{n} K\left(\frac{t - X_i}{d_k(t)}\right) \tag{1.12}$$

où $K(x)$ est une fonction noyau dont l'intégrale vaut 1, comme dans la section 1.4.4.

1.4.6 La méthode à noyau variable (the variable kernel method)

Cette méthode est introduite par [BMP77]. Elle est en relation avec la méthode des *k-ppv* et représente une autre méthode qui adapte la quantité de lissage de la densité locale des données. L'estimation est construite d'une manière similaire à celle de l'estimation classique par noyau, mais le paramètre de lissage s'adapte localement au point d'estimation et aux données. Soit K une fonction noyau et k un entier positif. Nous définissons la distance $d_{j,k}$ du point X_j au k^{me} plus proche point dans l'ensemble des autres $n-1$ points. L'estimateur à noyau variable dont

le paramètre de lissage est h est défini par

$$\widehat{f}(t) = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{hd_{j,k}}K\left(\frac{t-X_i}{hd_{j,k}}\right) \tag{1.13}$$

La largeur de la fenêtre du noyau placé au point X_j est proportionnelle à $d_{j,k}$. Alors dans les régions où les données sont denses, il y aura plusieurs noyaux associés ensemble. Pour chaque valeur de k fixée, le degré de lissage global dépendra du paramètre h.

Dans l'équation 1.12, la largeur de la fenêtre utilisée pour construire l'estimation au point t dépend des distances du point t aux autres points ; par contre dans l'équation 1.13 les largeurs de la fenêtre sont indépendantes du point t dans lequel la densité est en train d'être estimée et dépend uniquement des distances entre les données.

[Sil86] détaille davantage le principe de cette méthode ainsi que comment automatiser le choix du paramètre de lissage.

1.4.7 Généralisation pour le cas multivarié

Dans le cas de données multivariées, les méthodes paramétriques multivariées sont obtenus par généralisation de ces méthodes dans le cas univarié. Par exemple, soit $X = \{x_t\}_{t=1}^{n}$ un ensemble de données de dimension d. L'estimation par noyau multivarié est

$$\widehat{f}(x) = \frac{1}{nh^d}\sum_{t=1}^{n}K\left(\frac{x-x_t}{h}\right). \tag{1.14}$$

avec

$$\int_{\Re^d} K(x)dx = 1$$

Le candidat évident est le noyau gaussien multivarié défini par :

$$K(u) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{\|u\|^2}{2}\right] \tag{1.15}$$

1.5 Problèmes des espaces de grande dimension

Nous nous intéressons dans ce travail au cadre où le problème traité est de grande dimension, autrement dit, le nombre n de variables considérés est très grand. Dans ce cadre, les données disponibles sont le plus souvent de taille très

petite en comparaison avec la dimension n du domaine. Ce cas est le plus fréquent en réalité. Ce problème de grande dimension a été observé dans différents domaines :

Imagerie : Une base de données d'images peut être vue comme une matrice de taille $N \times n$, où N est le nombre d'images et n est le nombre de variables du domaine. Si une image est de taille $p \times p$ alors $n = p^2$. Certains travaux [CB06] ont étés amené à traiter des masses de données de taille 10 To d'images multi-angulaires en dimension 1000. Actuellement, de nouveaux capteurs sont capables de prendre des images de plus grande qualité. Ils n'utilisent plus uniquement trois couleurs pour représenter un pixel, mais plutôt tout le spectre avec plusieurs milliers de bandes fréquentielles. Une image n'est plus alors une matrice de $p \times p$ pixels, mais cela est multiplié par S le nombre de bandes spectrales pour chaque pixel. Ce qui fait exploser le nombre de variables à traiter.

Bioinformatique Dans le domaine de la bio-informatique, les données représentent généralement les différents gènes d'un chromosome. L'objectif dans ce cadre est d'apprendre quels sont les gènes mis en cause dans les maladies génétiques complexes. La chute des coûts de génotypage permet désormais la génération de centaines de milliers de marqueurs génétiques couvrant entièrement le génome, d'ou un grand nombre de variables. La taille des données disponibles pour ce type de problèmes est généralement trop faible devant la dimension. Par exemple, [MSL10] a été amené à traiter des données de taille 2000 échantillons pour une dimension de 10^5 variables, [EdAL10] a traité des données de taille 8 échantillons pour 7313 variables.

Dans le monde classique, les outils qui traitent ce genre de problèmes de grande dimension supposent qu'il faut traiter le problème en recherchant les variables les plus intéressantes et les mesurer en priorité, en utilisant par exemple de la connaissance scientifique. Cela n'est pas applicable dans la réalité. En effet, la théorie de ces méthodologies est basée sur l'hypothèse $n < N$, et $N \to \infty$. Plusieurs résultats ont utilisé des outils de l'algèbre linéaire et la théorie des groupes pour développer des distributions exactes pour des observations. Touts ces résultats échouent si $n > N$. Le pire est que ces outils supposent une situation asymptotique où $N \to \infty$ et n est fixé, ce qui contredit la réalité où il y a tendance à avoir n qui tend vers l'infini et N reste fixe et très faible.

Le cas $n > N$ n'est pas anormal, mais il s'agit du cas générique dans la réalité. Dans plusieurs types d'évènements, on peut avoir un grand nombre de quantités mesurables (variables) qui quantifient l'évènement, et un nombre relativement très limité d'instances de cet évènement. Par exemple, il est possible de disposer d'un grand nombre de gènes mais le nombre de malades portant une maladie génétique précise peut être relativement faible.

Le terme de « fléau de la dimension » (ou en anglais *curse of dimensionality*) est utilisé par [Bel57] pour décrire les problèmes causés par l'augmentation de la dimension de l'espace. Bellman a utilisé ce terme « fléau de la dimension » pour parler de la difficulté d'optimiser une fonction par une recherche exhaustive de l'*optimum* dans un espace discrétisé a travers un exemple : il faudra effectuer 10^{20} évaluations de la fonction pour rechercher l'optimum dans un cube gradué par une grille de pas 1/10 et de dimension 20. Silverman [Sil86] a également observé ce phénomène dans le cadre de l'approximation d'une distribution gaussienne quelconque avec des noyaux gaussiens fixés. Ses résultats montrent que le nombre N d'observations nécessaires à cette tâche avec une erreur maximale de 10% croît exponentiellement avec la dimension et peut être approché par la relation suivante :

$$\log_{10} N(n) \approx 0.6(n - 0.25)$$

Le phénomène de *"l'espace vide"* (dit aussi *espace creux*) est un autre problème dans les espaces de grandes dimensions. Scott et Thompson [ST83] ont montré l'exemple du "volume de la boule" qu'un espace de dimension n est presque vide puisque la très grande majorité des points se situe aux alentours d'un espace de dimension $n - 1$. Certains auteurs utilisent le phénomène de l'espace creux pour définir la limite entre les espaces de petite dimension et ceux de grande dimension. Certains travaux ont considéré que les espaces dont la dimension est plus grande que 5 sont des espaces de grande dimension. Plus simplement, certains pensent que si la dimension d'un espace est plus grande que 3, alors c'est un espace de grande dimension puisque l'humain ne peut pas naturellement se le représenter, alors que dire lorsque $n = 10^5$?.

Le phénomène de l'espace vide est de plus en plus important lorsque la dimension du problème augmente. En effet, dans les problèmes réels, les bases de données peuvent ne pas être suffisamment grandes et riches pour contenir toutes les configurations possibles des variables. L'absence de ces configurations (modélisé par des espaces vides) entraînera, dans le cas des estimateur MV, une probabi-

lité correspondante nulle comme si ces configurations étaient impossibles, ce qui n'est pas toujours vrai. Cela entraîne une mauvaise estimation des paramètres du problème à partir des données disponibles d'ou un résultat loin de la réalité. Le modèle estimé ne reflète pas le modèle réel ayant généré les observations et l'erreur engendrée par cette estimation est importante vue que le nombre d'échantillons disponible n'est pas suffisant pour une meilleure estimation. Par exemple, dans le domaine de la classification, les méthodes génératives, dont le but est de modéliser la densité de chacune des classes, souffrent de la mauvaise estimation des matrices de covariance a partir de données relativement de quantité très faible en comparaison avec la dimension du problème. En effet, l'estimation des matrices de covariances correspond à l'estimation d'un nombre de paramètres qui croît avec le carré de la dimension. Si le nombre d'observations est trop petit devant la dimension de l'espace, alors les estimations de ces matrices seront singulières et leur inversion sera impossible. Ce mauvais conditionnement des matrices de covariance entraînera lors de leur inversion un important biais sur le calcul de la règle de décision et ensuite une importante erreur de prédiction.

Les méthode d'estimation non paramétrique de la densité souffrent aussi de ce même problème de l'espace vide. En effet, par exemple, si X est de dimension 8, et nous utilisons un estimateur de type histogramme avec 10 boites par dimension, le nombre total de boites est alors 10^8. Si les données disponibles sont de faible taille, alors la majorité des boites seront vides et ainsi l'estimation par MV de la probabilité correspondante sera nulle. Dans les espaces de grande dimension, la notion de "proximité" devient floue, il faut alors faire attention en choisissant le paramètre h (largeur de la boite).

De plus, l'augmentation du nombre de variables du problème entraîne une augmentation des paramètres nécessaires pour représenter le modèle global, et par conséquent, elle entraîne le phénomène du «sur-apprentissage» (connu sous le nom *overfitting*) . En effet, sans précaution particulière, un modèle complexe qui dépend d'un grand nombre de paramètres va parfaitement "épouser" la forme des données d'apprentissage et du coup devenir très dépendant de ces données. Il aura sur-appris la forme de ces données. Dans le cas où les données d'apprentissage ne seraient pas représentatives du processus qui les a générées (très peu de données par exemple), le modèle estimé ne pourra pas être efficace pour traiter de nouvelles données, différentes des données d'apprentissage. Si le modèle estimé est trop simple, il ne pourra ni être efficace sur les données d'apprentissage ni sur les données de test. On parle dans ce cas de «sous-apprentissage». Il est donc

important de trouver le bon degré de complexité pour le modèle qui représentera les données.

D'autre part, le problème du « fléau de la dimension » engendre de nombreux problèmes pratiques lors de la manipulation des grandes dimensions. Citons essentiellement les difficultés en terme de ressources nécessaires pour pouvoir effectuer la recherche de l'optimum d'une fonction. Certaines méthodes numériques, dont la complexité est élevée en fonction de la dimension du problème, nécessitent un temps de calcul beaucoup plus important lorsque le nombre de variables de la fonction augmente. L'algorithme EM appliqué aux mélanges de gaussiennes souffre aussi d'une lenteur de sa convergence lorsque la dimension du problème est importante. D'autres ressources telles que les ressources en mémoire dépendent directement du nombre de variables.

De manière générale, plusieurs méthodologies ont été appliqués pour l'analyse et le traitement des données de grande dimension. Ces méthodes sont basées généralement sur la réduction dimensionnelle des données de façon globale, la régularisation des estimations des matrices de covariance ou l'utilisation d'un modèle qui soit parcimonieux [CB06] comme les modèles graphiques probabilistes étudiés en chapitre 2.

En ce qui concerne le problème des données qui sont de taille très faible devant la dimension du domaine, des techniques ont été utilisées pour améliorer la qualité de l'estimation dans de tels contextes, citons le Boosting et le Bagging qui seront présentés dans le chapitre 3.

1.6 Conclusion

Nous avons introduit dans ce chapitre la notion de l'estimation de densité à partir de données d'observation. Nous avons commencé par présenter le principe général de l'estimation de densité, ensuite nous avons présenté dans la section 1.3 les différentes méthodes paramétriques ainsi que quelques exemples dans le cadre univarié et multivarié. La section 1.4 a servi à présenter les approches non-paramétriques d'estimation de densité. Dans la dernière section 1.5, nous avons discuté les problèmes liés aux espaces de grande dimension auxquels nous sommes de plus en plus confrontés.

Le grand nombre de paramètres à estimer que nécessitent les lois classiques est à l'origine de tous ces problèmes. De plus, le nombre de données disponibles qui est généralement faible devant la dimension de l'espace entraîne une mauvaise

estimation de ces paramètres. Il est alors judicieux d'essayer de réduire le nombre de paramètres nécessaires pour définir une densité de probabilité en factorisant sa loi jointe et en utilisant les dépendances et les indépendances entre les variables, ceque les modèles graphiques permettent de faire.

Chapitre 2

Les modèles graphiques probabilistes

2.1 Introduction

Les modèles graphiques probabilistes (MGP) sont le mariage entre la théorie des probabilités et la théorie des graphes. Ils fournissent un outil pour la gestion simultanée de deux problèmes "l'incertain" et la "complexité". Les modèles graphiques bénéficient non seulement des avantages des modèles probabilistes, mais, de plus, ils présentent des avantages supplémentaires liés à leur représentation graphique. En effet, le coté graphique de ces modèles fournit à la fois une interface intuitive par laquelle on peut modéliser un grand nombre de variables aléatoires en représentant les indépendances conditionnelles entre elles ainsi qu'une structure de données qui se prête naturellement à la conception d'algorithmes généraux et efficaces. Le coté probabiliste des modèles graphiques permet de factoriser une distribution de la loi jointe qui relie les variables. Le coté graphique permet en plus de visualiser l'architecture du modèle complexe représentant le problème ainsi que les relations entre les différentes parties du modèle (variables et ensemble de variables).

Selon la nature de la structure du graphe, les MGP se divisent en trois classes : les DAG (où les arcs du graphe sont orientés), les Champs de Markov (où les arcs sont non orientés) et les Chains Graphs (où certains arcs sont orientés et d'autres sont non orientés).

Ce chapitre est consacré à l'étude des MGP de différents points de vues. Nous commençons dans la section 2.2 par introduire le principe général de ces modèles

graphiques en mettant l'accent principalement sur la factorisation de la loi jointe ainsi que les avantages de ces modèles par rapport à d'autres méthodes. Nous étudions aussi dans cette même partie les méthodes d'inférence en général et l'apprentissage des paramètres à partir de données d'observation.

Nous détaillons ensuite la notion d'apprentissage de structure et des paramètres et les méthodes de simulation de données dans chacune des classes des MGP. Nous nous intéresserons plus particulièrement aux modèles graphiques dirigés (DAG) où les variables sont discrètes (les Réseaux Bayésiens) (Section 2.5.2).

Nous étudierons ensuite dans la section 2.8 le cas particulier des MGP dont la structure du graphe est simple. Ce type de graphes (les arbres) sera notre outil pour les chapitres suivants.

2.2 Principes généraux

2.2.1 Définition

Un modèle graphique de dépendances est une représentation composée d'une paire d'ensembles $G = (V, E)$, où V est un ensemble fini de noeuds (*vertices*) et l'ensemble E des arcs (*edges*) est un sous ensemble de $V \times V$ de paires ordonnées de noeuds distincts. Un graphe est alors une structure simple ne contenant ni des arcs multiples ni des cycles. Un tel modèle graphique est alors un modèle de dépendances défini par la définition 2.

Définition 2 (*Graphe de dépendances*)

Un graphe de dépendance $G = (V, E)$, orienté ou non, est composé d'un ensemble de noeuds V, représentant des variables aléatoires et un ensemble d'arêtes E tel que l'absence d'une arête entre les noeuds u et v représente une relation d'indépendance conditionnelle (définition 3) entre les variables aléatoires associées à ces deux noeuds.

Définition 3 (*Indépendance conditionnelle*)

Soit A, B et C trois ensembles de variables aléatoires. A et B sont indépendants conditionnellement à C si et seulement si :

$$P(A, B|C) = P(A|C)P(B|C)$$

et on note $A \perp B|C$.

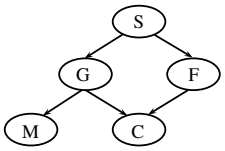
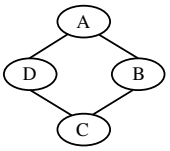
	graphe dirigé	graphe non dirigé
Représentation graphique		
Indépendances	$(F \perp H \mid S)$ $(C \perp S \mid F, H)$ $(M \perp H, C \mid F)$ $(M \perp C \mid F)$	$(A \perp C \mid B, D)$ $(B \perp D \mid A, C)$
Factorisation	$P(S,F,H,C,M) = P(S)P(F\|S)$ $P(H\|S)P(C\|F,H)P(M\|F)$	$P(A,B,C,D) = \frac{1}{Z}\phi_1(A,B)$ $\phi_2(B,C)\phi_3(C,D)\phi_4(A,D)$
	(a)	(b)

Figure 2.1. Indépendances conditionnelles et factorisation de la loi jointe par des modèles graphiques probabilistes : haut - la représentation graphique ; milieu - les indépendances induites par la structure du graphe ; bas - la factorisation induite par la structure du graphe. (a) un exemple de DAG. (b) un exemple de champ de Markov.

D'après cette définition des MGP, il parait bien le lien que fait ce type de modèles entre les deux formalismes : la théorie des probabilités (indépendances conditionnelles) et la théorie des graphes.

Un graphe ne contenant que des arcs dirigés est appelé *graphe dirigé*, et si tous les arcs sont non dirigés alors le graphe est dit *non dirigé*. Il peut exister aussi un graphe qui contient à la fois des arcs dirigés et d'autres non dirigés, dans ce cas le graphe est dit *mixte ou semi-dirigé*.

Les modèles graphiques probabilistes se divisent alors en plusieurs classes. Les classes les plus connues sont (1) les *DAG* où le graphe est dirigé, (2) les *champs de Markov* où le graphe est non dirigé, et (3) les *Graphes mixtes ou Chains graphs* où le graphe contient des arcs dirigés et d'autres non dirigés.

2.2.2 *Comment un modèle graphique peut représenter une distribution de loi jointe d'une façon compacte ?*

Les modèles graphiques probabilistes représentent graphiquement les indépendances. L'ensemble V de noeuds est en bijection avec un ensemble X de variables aléatoires. Un noeud représente une variable. La topologie de ce graphe est utilisée pour représenter les dépendances entre les variables. Ainsi, un arc entre deux noeuds indique une relation de dépendance entre les deux variables associées et l'absence d'un arc indique une relation d'indépendance. Un modèle graphique probabiliste est capable de représenter des indépendances conditionnelles au sein d'un groupe de variables (voir définition 3) grâce à la notion de séparation, ceci se traduit par les propriétés de Markov au sein du graphe lui même (pour plus de détails sur les propriétés de Markov pour les modèles graphiques, voir l'annexe B inspirée de [BK02] et [NWL+04]).

De façon générale, grâce aux indépendances entre les variables que les modèles graphiques probabilistes sont capables de représenter, ces derniers permettent d'écrire la fonction densité f de probabilité jointe comme le produit de fonctions élémentaires f_i définies chacune sur un sous ensemble de variables (Equation 2.1). La figure 2.1 montre un exemple de factorisation de la probabilité jointe en fonctions élémentaires dépendant de sous ensembles de variables pour un exemple de DAG et un autre pour un graphe non dirigé.

$$f(X_1, \ldots, X_n) = \prod_i f_i(X_1, \ldots, X_n) \tag{2.1}$$

Les modèles graphiques probabilistes fournissent alors une représentation compacte de la distribution de probabilité jointe. En effet, si nous avons n variables aléatoires binaires, une représentation complète de la distribution jointe $P(X_1, \ldots, X_n)$ nécessite $O(2^n)$ paramètres, tandis qu'un modèle graphique peut utiliser un nombre exponentiellement moins, cela dépend de la complexité de sa structure (les dépendances qu'il représente). Alors, si le problème à traiter est de grande dimension (n très grand), la factorisation de la loi jointe par le moyen des modèles graphiques s'avère un bon moyen pour réduire la complexité du problème. Les distributions de probabilités jointes sont définies en faisant le produit de fonctions définies sur seulement les sous-ensembles de noeuds connectés.

Nous détaillerons dans la suite l'équation 2.1 dans chacun des cas des modèles graphiques dirigés, non dirigés et partiellement dirigés en donnant la forme exacte des fonctions élémentaires f_i.

Dans la figure 2.1 issue de [KF09], la figure 2.1(a) représente un graphe dirigé formé par des noeuds (représentant les variables aléatoires) et des arcs orientés (représentant les relations entre les variables) ainsi que les indépendances conditionnelles qu'il représente et la factorisation compacte de la distribution qu'il encode. De même, la figure 2.1(b) est relative à un graphe non dirigé. Les deux représentations fournissent la dualité des indépendances et factorisation, mais elles diffèrent dans l'ensemble des indépendances qu'elles peuvent encoder et de la factorisation de la distribution qu'elles induisent.

2.3 Inférence dans les modèles graphiques probabilistes

En général, le problème de l'inférence est de répondre à une requête de la forme Q = "quelle est la probabilité de la variable v dont la valeur est x_v sachant que les valeurs des variables dans le sous ensemble $X_v \subseteq X$ sont connues ?". Les variables dans X_v et leurs valeurs observées sont considérés comme *evidence* et notée par ϵ. Alors, l'inférence consiste à calculer la probabilité $P(x = x_v|\epsilon)$ dans le modèle donné.

Grâce aux indépendances conditionnelles associées à la factorisation de la distribution jointe, on peut souvent effectuer l'inférence dans ces modèles même complexes, plus facilement qu'en utiliser directement la loi jointe.

[Coo90] a montré que le problème de calcul des probabilités a posteriori est NP-difficile. L'inférence dans les modèles graphiques en général est alors un problème NP-difficile, mais peut être effectuée efficacement par le moyen des algorithmes d'inférence exacte lorsque le modèle est simple. La complexité de ces algorithmes dépend alors de la structure du graphe. Lorsque la dimension du graphe augmente, l'inférence peut être difficile et nécessitera un long temps de calcul. Dans ce cas on a recours à des méthodes approximatives qui se divisent principalement en deux catégories [MSA+06] : La première catégorie groupe les techniques basées sur l'échantillonnage de type Monte Carlo [Hen86, GRS96], tandis que la deuxième catégorie concerne les méthodes variationnelles [JGJS98].

2.3.1 Élimination de variables

Un premier algorithme d'inférence exacte appelé *élimination de variables* a été initialement proposé par [ZP94]. Son principe consiste en la suppression des

variables une à une par marginalisation. Il a été généralisé dans [Dec98] pour donner naissance à la méthode dite *bucket elimination*. Un ordre des variables doit être donné en entrée et sera alors l'ordre d'élimination des variables. La quantité de calculs nécessaire dépend de cet ordre de suppression des variables puisqu'il influe sur la taille des facteurs futurs : différents ordres d'élimination produisent différents facteurs.

Trouver le meilleur ordre est équivalent au problème de trouver l'arbre de plus petite largeur dans le réseau [Dec98], ce qui est un problème NP-difficile [AP89].

Cet algorithme d'élimination de variables prend en entrée un ensemble de facteurs, alors il ne dépend pas de la nature (dirigé, non dirigé ou semi-dirigé) du graphe qui a généré les facteurs [KF09].

2.3.2 Messages locaux

Dans [Pea85], Pearl a introduit un algorithme d'inférence exacte pour les structures simples appelées *polyarbres* (singly connected tree). Un polyarbre est un graphe dont la structure non dirigée ne présente pas de cycles. Dans un tel réseau, il existe toujours un seul chemin (non dirigé) entre n'importe quelles deux variables u et v. L'algorithme de Pearl, connu aussi sous le nom de *Message Passing*, procède par des calculs locaux, en chaque noeud du graphe. De proche en proche, chaque noeud communique à ses voisins les informations qu'il a collectées (les probabilités conditionnelles connues), jusqu'à ce que tout noeud puisse mettre à jour sa probabilité marginale et qu'un équilibre soit atteint, en un nombre fini d'étapes. La complexité en mémoire de cette méthode est polynomiale mais ne s'applique que sur les MGP simples.

Deux approches principales ont été proposées pour généraliser cet algorithme de *Messages locaux* pour les réseaux quelconques (présence de cycles). Ces deux approches sont le *Cutset conditionning* et *l'arbre de jonction*.

2.3.3 Loop Cutset conditionning

La méthode du *Loop Cutset conditionning* [Pea91] consiste à instancier un certain nombre de variables appelé *loop cutset*, de manière à ce que le graphe restant forme un arbre. Ensuite, on procède à une propagation par messages sur cet arbre. Puis une nouvelle instanciation est choisie, et on réitère ce processus jusqu'à ce que toutes les instanciations possibles aient été utilisées. Les résultats

sont moyennés en les pondérant par leur probabilité *a priori*. La complexité de cet algorithme augmente exponentiellement en fonction de la taille du *loop cutset*. Il est alors important de minimiser la taille du *loop cutset*, qui est un problème NP-difficile.

2.3.4 Arbre de jonction

Cette méthode est introduite par [LS88] et [JLO90], Elle s'appelle aussi *la méthode JLO* pour les noms de ses auteurs (Jensen, Lauritzen et Olesen). Elle est utilisable dans le cas des modèles graphiques quelconques.

Cet algorithme est une généralisation de l'algorithme *Message locaux* pour un graphe quelconque. Il consiste à transformer le graphe en un hyper-arbre dont chaque noeud est une clique. L'arbre de cliques ainsi construit est appelé arbre de jonction. Les cliques constituant les noeuds de l'arbre ne sont pas celles du graphe original, mais celles d'un graphe augmenté obtenu après les étapes suivantes :

- **Moralisation du graphe** : Cette étape consiste à *marier* deux à deux les parents de chaque variable en les reliant par un arc supplémentaire non dirigé. Ensuite, le graphe résultant est complètement transformé en graphe non dirigé en enlevant les directions de chaque arc. On obtient ainsi un graphe non dirigé dit *moralisé*.
- **Triangulation du graphe moral** : Cette étape consiste à ajouter des arcs entres les variables jusqu'a obtention d'un graphe triangulé. L'algorithme de triangulation opère de la manière suivante. Un graphe est triangulé si et seulement si l'ensemble de ses noeuds peuvent être éliminés. Un noeud peut être éliminé si tous ses voisins sont connectés deux à deux. Donc un noeud peut être éliminé s'il appartient à une clique dans le graphe. Une telle clique forme un noeud pour le futur arbre de jonction qui est en train d'être construit. Il est à noter qu'un graphe triangulé pour un graphe moral n'est pas unique. Cela dépend de l'ordre des connexions ajoutées au graphe moral.
- **Construction de l'arbre de jonction** : Un arbre de jonction est une structure arborescente ayant l'ensemble des cliques d'un graphe triangulé comme ensemble de noeuds. Il est possible de construire l'arbre de jonction au moment de la triangulation. Les variables de l'arbre de jonction sont des groupes de variables du graphe d'origine.Tout comme le graphe triangulé n'est pas unique, l'arbre de jonction ne l'est pas non plus.

La loi jointe de cet arbre de jonction peut être écrite comme un produit de

facteurs sur les cliques, appelés *potentiels de cliques*.

Une fois l'arbre de Jonction construit, il suffit de faire l'inférence dans cet arbre en utilisant l'algorithme du *Message locaux* et en respectant son protocole : une clique ne peut envoyer de message à son voisin que si elle a reçu tout les message des autres voisins. Enfin, il faut transformer les potentiels de cliques en lois conditionnelles mises à jour.

La complexité de cette méthode est exponentielle en fonction de la taille de la plus grande clique de l'arbre de jonction [LS88]. Cet algorithme d'arbre de Jonction n'est alors plus pratique dès que la largeur de l'arbre devient importante.

2.4 Apprentissage des paramètres des modèles graphiques probabilistes

Nous avons vu que les MGP permettent de factoriser la distribution de probabilité jointe (Equation 2.1) en l'écrivant sous la forme de produit de fonctions élémentaires.

Soit un MGP de structure G définie par un ensemble complet de n variables $(X_1, \ldots, X_n)$ et soit D un ensemble de N échantillons *iid*. La densité de la probabilité jointe s'écrit alors comme suit :

$$P(D|G,\theta) = \prod_k P_k(D|G,\theta_k) \tag{2.2}$$

Nous supposons que la structure graphique du MGP. L'apprentissage des paramètres d'un MGP à partir de données complètes peut être effectué d'une façon automatique en utilisant l'une des méthodes paramétriques : la méthode du Maximum de Vraisemblance (section 1.3.2) ou la méthode bayésienne du maximum a posteriori (section 1.3.3). Dans le cas où le graphe est dirigé, en faisant l'hypothèse classique d'indépendances des paramètres, les probabilités conditionnelles de chaque variable sont calculées indépendamment des autres variables grâce à l'indépendance des paramètres. Donc, on parle de calcul local du maximum de vraisemblance (ou MAP). Dans le cas de graphe non dirigé, cette décomposition locale du maximum de vraisemblance n'est plus possible. Il faut alors avoir recours à des méthodes itératives (par exemple la méthode IPF *iterative proportional fitting* proposée par [DS40]).

2.5 Les DAG : les graphes orientés sans circuits

2.5.1 Approche générale

Un DAG $B = (G, \theta)$ est défini par

- $G = (X, E)$, un graphe orienté sans circuit où X est l'ensemble des noeuds (ou sommets) associés à un ensemble de variables aléatoires $X = (X_1, \ldots, X_n)$ et où E est l'ensemble des arcs,
- $\theta = \{\theta_i\} = f(X_i|Pa(X_i))$, ensemble des fonctions de densité de probabilité de chaque noeud X_i conditionnellement à l'état de ses parents $Pa(X_i)$ dans G.

Les DAG permettent de représenter de manière compacte la distribution de probabilité jointe sous la forme d'un produit de fonctions élémentaires définies sur un sous ensemble de variables :

$$f(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} f(X_i|Pa(X_i)) \tag{2.3}$$

où $f(X_i|Pa(X_i))$ représentent les fonctions élémentaires f_i de l'équation 2.1.

Une fois la structure du DAG fixée, l'apprentissage des paramètres du DAG à partir de données peut être effectué efficacement par l'une des méthodes paramétriques citée dans 1.3 : la méthode du maximum de vraisemblance ou la méthode du maximum a posteriori.

Apprendre la structure d'un modèle graphique à partir de données n'est cependant pas une tâche facile, en particulier à cause de la taille de l'espace de recherche. [Rob77] a montré que le nombre $r(n)$ de structures différentes pour n noeuds est donné par la formule de récurrence de l'équation 2.4. Comme cette fonction est super-exponentielle, il devient très difficile (non réalisable) d'effectuer un parcours exhaustif de l'espace de recherche en un temps raisonnable dès que le nombre de variables augmente. Il est donc nécessaire de recourir à des heuristiques de recherche dans l'espace des DAG. Ces heuristiques de recherche doivent être principalement dépendantes du but de cet apprentissage, par exemple pour la découverte de connaissances ou pour l'estimation de densité.

$$r(n) = \sum_{i=1}^{n} (-1)^{i+1} \begin{pmatrix} n \\ i \end{pmatrix} 2^{i(n-1)} r(n-i) = n^{2^{O(n)}} \tag{2.4}$$

2.5.2 Le cas discret : les réseaux bayésiens (RB)

Un DAG dont toutes les variables sont discrètes est appelé un réseau bayésien.

2.5.2.1 Apprentissage des paramètres

Dans le cas discret, un paramètre $\theta_i = P(X_i|Pa(X_i))$ est un tableau contenant l'ensemble des probabilités de la variable X_i pour chacune de ses valeurs possibles sachant chacune des valeurs prises par l'ensemble de ses parents $Pa(X_i)$ dans le graphe G.

Soit un ensemble de données D de taille N. L'apprentissage des paramètres d'un RB à partir de ces données se fait de la même façon que pour les lois multinomiales décrites dans la section 1.3.4.2.

Dans le cas de l'estimation par maximum de vraisemblance, l'estimation $\hat{\theta}^{MV}_{i,j,k}$ de la probabilité conditionnelle de X_i sachant ses parents $Pa(X_i)$ est donnée par l'équation suivantes :

$$\hat{P}(X_i = x_k|Pa(X_i) = x_j) = \hat{\theta}^{MV}_{i,j,k} = \frac{N_{i,j,k}}{\sum_k N_{i,j,k}}$$

où $N_{i,j,k}$ est le nombre d'évènements dans la base de données pour lesquels la variable X_i est dans l'état x_k et ses parents sont dans la configuration x_j.

Dans le cas de l'estimation par Maximum a posteriori, soit une distribution de Dirichlet de paramètres $\alpha_{i,j,k}$ associée à la loi a priori $P(X_i = x_k|Pa(X_i) = x_j)$.

L'estimation Maximum a posteriori $\hat{\theta}^{MV}_{i,j,k}$ de la probabilité conditionnelle de X_i sachant ses parents $Pa(X_i)$ est données par l'équation suivantes :

$$\hat{P}(X_i = x_k|Pa(X_i) = x_j) = \hat{\theta}^{MAP}_{i,j,k} = \frac{N_{i,j,k} + \alpha_{i,j,k} - 1}{\sum_k (N_{i,j,k} + \alpha_{i,j,k} - 1)}$$

2.5.2.2 Apprentissage de structure

Dans le cas où notre base de données est complète, c'est-à-dire toutes les mesures sont complètes, deux familles d'approches ont été proposées pour faire de l'apprentissage de structure des RB : la première est basée sur la recherche des indépendances conditionnelles, tandis que la deuxième utilise une fonction de score. Une troisième famille d'algorithmes a vu le jour pour apprendre la structure optimale d'un RB : les méthodes hybrides.

Méthodes de recherche d'indépendances conditionnelles

La première famille (algorithmes IC [PV91], PC [SGS93], FCI [SGS00] etc.) consiste à déterminer dans un premier temps un graphe non orienté en tenant compte des différentes indépendances conditionnelles qui existent entre les variables de ce graphe, puis à orienter ce graphe pour obtenir un réseau bayésien. Ces algorithmes sont peu efficaces dans le cas de problèmes de grande taille puisque la détermination de ces indépendances est exponentielle en fonction du nombre de variables. Les méthodes de recherche d'indépendances conditionnelles ont donc une complexité élevée.

Méthodes à base de scores

La deuxième approche consiste à parcourir tous les graphes possibles, associer un score à chaque graphe, puis choisir le graphe ayant le score le plus élevé. Plusieurs fonctions de score peuvent être utilisées, nous citons le score BD (Bayesian Dirichlet), le score BDe (Bayesian Dirichlet Equivalent), le score $BDeu$, le score BD_γ (Bayesian Dirichlet généralisé), l'entropie conditionnelle, le score AIC (Akaike Information Criterion), le score BIC et la longueur de description minimale (MDL) (voir [NWL+07] pour plus de détails sur ces scores).

Ces méthodes ne sont pas simples, principalement à cause de la taille super-exponentielle de l'espace de recherche en fonction du nombre de variables.

En supposant que nous disposions d'une méthode qui parcourt l'espace des DAG en effectuant des opérations du type *ajout* ou *suppression* d'arcs, il est nécessaire de réduire le nombre de calculs utilisés pour l'évaluation du score. Pour cela, posséder un score calculable localement, permet de n'estimer que la variation de ce score entre deux structures voisines, au lieu de le recalculer entièrement pour la nouvelle structure. Un tel score est dit *score décomposable* et est défini comme suit.

Définition 4 (*Score décomposable, score local*)

Un score S *est dit* ***décomposable*** *s'il peut être écrit comme une somme ou un produit de* ***scores locaux*** *dont chacune n'est fonction que seulement d'un noeud et de l'ensemble de ses parents. Si* n *est le nombre de noeuds du graphe* G*, le score doit s'écrire sous l'une des formes suivantes :*

$$S(G) = \sum_{i=1}^{n} s(X_i, Pa(X_i)) \quad ou \quad S(G) = \prod_{i=1}^{n} s(X_i, Pa(X_i))$$

Définition 5 (*Score équivalent*)

Un score qui associe une même valeur à deux graphes équivalents (voir annexe B) est dit ***équivalent****. C'est-à-dire, si G_1 et G_2 sont deux graphes équivalents au sens de Markov, et S un score équivalent, alors $S(G_1) = S(G_2)$.*

Exemple de score décomposable : Le score BDeu

Le score BDeu définit par l'équation 2.5 est un score décomposable. Soit G un graphe et D un ensemble de données d'apprentissage.

$$ScoreBDeu(G, D) = P(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \tag{2.5}$$

avec r_i est la taille du noeud X_i dans G, q_i est le nombre de configurations possibles des parents de X_i, N_{ij} est le nombre d'occurrences de $\{X_i = x_i \; et \; X_j = x_j\}$ et les α_{ijk} sont des a priori décrits dans l'équation 2.6, où N' est un nombre d'exemples équivalent définis par l'utilisateur.

$$\alpha_{ijk} = \frac{N'}{r_i q_i} \tag{2.6}$$

Il est à noter que le score $BDeu$ est aussi un score équivalent.

D'autres idées ont été proposées pour résoudre ce problème de l'espace de recherche. Une première consiste à remplacer l'espace de recherche (espace des DAG) par un espace plus petit, l'espace des arbres. Cette idée consiste à rechercher l'arbre optimal en utilisant une notion classique dans en recherche opérationnelle, l'arbre de recouvrement maximal (MWST ou Maximal Weight Spanning Tree) : c'est l'arbre qui relie tous les noeuds en maximisant un critère (un score défini pour tous les arcs possibles entre les variables). Cette méthode sera présentée et discutée ultérieurement dans la section 2.8.

Une deuxième idée consiste à ordonner les noeuds pour limiter la recherche des parents possibles pour chaque noeud (algorithme K2 [CH92]). Une troisième consiste à faire une recherche gloutonne dans l'espace des réseaux bayésiens (algorithme GS [Chi03]).

Notons qu'il existe un autre espace dans lequel nous pouvons rechercher notre graphe : l'espace des représentants des classes d'équivalence de Markov (voir annexe B). Cet espace est équivalent à l'espace des réseaux bayésiens sauf qu'il ne contient que des représentants des classes d'équivalence de Markov, donc un peu

plus petit.

Méthodes hybrides

Divers travaux ont proposé de combiner l'apprentissage par recherche d'indépendances conditionnelles et celui à base de scores afin de profiter des avantages de chacune des deux approches. Cela a donné le jour à des méthodes hybrides. La première catégorie d'approches hybride va profiter des avantages des méthodes à base de score pour aider les algorithmes d'apprentissage de structure par recherche d'indépendance conditionnelle. Citons l'algorithme proposé par [CGK+02].

D'autres approches, symétriques aux précédentes, vont utiliser les tests d'indépendance conditionnelle pour générer un ordre d'énumération des variables qui guidera les algorithmes d'apprentissage de structure à base de score vers la solution optimale. Citons l'algorithme MMHC (Max-Min Hill Climbing) proposé par [TBA06], les algorithmes proposés par [SV93] et [LRS04]. Un autre algorithme hybride a été proposé par [KA09]. Il représente une nouvelle méthode hybride d'apprentissage de la structure des réseaux bayésiens combinant des techniques de data mining, d'apprentissage sous contraintes et à base de score.

2.5.3 Le cas continu : les réseaux bayésiens gaussiens

Un réseau bayésien gaussien est un modèle graphique qui représente la structure de dépendances d'une variable aléatoire normale multivariée en utilisant un graphe dirigé acyclique (DAG). Un réseau bayésien gaussien est alors un DAG dont les variables sont continues et ont une distribution de probabilité gaussienne linéaire.

Un résultat important est qu'un réseau bayésien gaussien est une représentation alternative de la classe des distributions gaussiennes multivariées. Un réseau bayésien gaussien définit alors une distribution jointe gaussienne multivariée et vice versa (voir [Nea04], chapitre 7).

Un réseau bayésien gaussien est défini comme suit : Si Pa_X est l'ensemble des parents de X, alors

$$x = w_X + \sum_{Z \in Pa_X} b_{XZ} z,$$

où W_X a une fonction de densité $N(w; 0, \sigma^2_{W_X})$, et W_X est indépendante de tout Z. La variable W_X représente l'incertain de la valeur de X sachant les valeurs des

parents de X. La fonction de densité conditionnelle de X est alors :

$$\rho(x|Pa_X) = N(x, \sum_{Z \in Pa_X} b_{XZ} z, \sigma^2_{W_X})$$

2.5.3.1 Apprentissage des paramètres

Comme il est possible de transformer un réseau bayésien gaussien en une distribution normale multivariée (voir [Nea04] pour plus de détails sur la transformation), il suffit d'effectuer cette transformation et utiliser la méthode décrite dans la section 1.3.5.2 pour faire l'apprentissage des paramètres du réseau bayésien gaussien. Un algorithme d'apprentissage des paramètres d'un réseau bayésien dont les variables sont continues est décrit dans le chapitre 7 de [Nea04].

De même que pour le cas discret, grâce à l'indépendance des paramètres, on peut estimer séparément les paramètres de chaque loi conditionnelle de chaque variables. [KF09] détaille comment le principe du MV peut être appliqué dans le cas des réseaux Bayésiens gaussiens.

2.5.3.2 Apprentissage de structure

L'apprentissage de structure d'un réseau bayésien gaussien à partir de données peut se faire d'une façon analogue à celle dans le cas discret. [Nea04] propose une méthode à base de score qui recherche dans l'espace des équivalences de Markov la structure de DAG ayant la probabilité a posteriori la plus élevée sachant les données. [GH94] discute l'apprentissage de structure de réseaux Bayésiens gaussiens à base de score et propose des adaptation de scores pour ce cas de modèle.

2.5.4 Le cas hybride : les réseaux conditionnels gaussiens

Un réseau bayésien qui contient à la fois des variables discrètes et d'autres continues est dit *Conditionnel Gaussien* (CG) [LW89a, Lau92]. Dans ce type de modèles, pour chaque instanciation i d'un ensemble de noeuds discrets Y la distribution des noeuds continus X est de la forme $f(x|Y = i) = N(x, \vec{\mu}(i), \Sigma(i))$.

Il est à noter que dans le type de modèles que nous considérons dans ce paragraphe, un noeud discret ne peut pas avoir de parents continus.

La probabilité conditionnelle d'un noeud discret X_i sachant ses parents $Pa(X_i)$ (qui ne peuvent être que discrets) est représentée par une table comme nous l'avons déjà vu pour le cas des réseaux bayésiens à variables discrètes.

Algorithme 1 Algorithme Forward Sampling in a Bayesian network ([KF09])

1: **Procédure** Forward-Sample(B {*Réseau Bayésien en* X})
2: Soit $X_1, \ldots, X_n$ un ordre topologique de X
3: **Pour** $i = 1$ to n **Faire**
4: $u_i \leftarrow x \langle Pa_{X_i} \rangle$ {*représente* Pa_{X_i} *dans* $x_1, \ldots, x_{i-1}$}
5: Échantillonner x_i à partir de $P(X_i|u_i)$
6: **Fin Pour**
7: **return** $(x_1, \ldots, x_n)$

La distribution conditionnelle d'un noeud continu X_i sachant ses parents $Pa(X_i)$ peut être spécifiée par une fonction gaussienne dont la moyenne est une fonction linéaire des parents et la covariance est fixée (voir [Mur98] pour plus de détails).

Dans ce type de modèles, l'inférence est un problème NP-difficile même dans le cas des polyarbres, [KF09] détaille les différentes difficultés rencontrées lorsque le modèle est constitué à la fois de variables discrètes et d'autres continues.

[MC98] propose deux méthodologies à base de score pour l'apprentissage de structure des réseaux bayésiens hybrides à partir de données. La première méthodologie consiste à discrétiser les données pour ensuite utiliser les algorithmes classiques applicables pour le cas discret. La deuxième méthodologie consiste à utiliser des réseaux de neurones comme estimateurs de probabilités qui seront ensuite utilisée pour calculer des scores.

2.5.5 Simulation de données selon la loi d'un DAG

Il est facile d'échantillonner de nouvelles données à partir d'un DAG du moment où tous ses paramètres sont connus. L'algorithme 1 pris de [KF09] montre qu'on peut échantillonner les noeuds selon un ordre topologique donné par le DAG. Un noeud est échantillonné à partir de la distribution définie par sa CPD (distribution de probabilité conditionnelle), sachant les instantiations de tous ses parents.

Dans le cas de où les variables sont discrètes, pour une distribution uniforme, il suffit d'utiliser un générateur de nombres pseudo-aléatoires. Les autres distributions nécessitent des techniques plus spécifiques. Une méthode présentée dans [KF09] est la suivante. Soit une distribution multinomiale $P(X)$ pour $Val(X) = \{x^1, \ldots, x^k\}$, définie par les paramètres $\theta_1, \ldots, \theta_k$. Le processus peut être effectué de la façon suivante : on génère un échantillon s uniformément dans

l'intervalle $[0, 1]$. Ensuite, on partitionne l'intervalle en k sous-intervalles : $[0, \theta_1[$, $[\theta_1, \theta_1 + \theta_2[$, ... ; de façon à ce que le i ème intervalle est $[\sum_{j=1}^{i-1} \theta_j, \sum_{j=1}^{i} \theta_j[$. Si s est dans le $i$ème intervalle, alors la valeur échantillonnée est x^i.

Cette approche est une solution générale pour générer des échantillons à partir de n'importe quelle CPD d'une variable discrète : étant donné un parent u, on peut générer la distribution conditionnelle $P(X|u)$ et échantillonner à partir de cette distribution de la même façon.

Ce processus d'échantillonnage de données nécessite une opération d'échantillonnage pour chaque variable aléatoire dans le réseau. Il faut alors utiliser pour chaque variable X une CPD en fonction de ses parents, cela peut se faire en un temps de calcul $O(|Pa_X|)$. Le processus d'échantillonnage présenté nécessitera alors un temps $O(log|Val(X)|)$. Si M est le nombre total d'échantillons générés, $n = |X|$, $p = \max_i |Pa_{X_i}|$, et $d = \max_i |Val(X_i)|$, le coût total est en $O(Mnp \log(d))$.

La génération d'échantillons à partir d'une distribution en présence d'une évidence e est un problème plus compliqué à résoudre. En général, on s'intéresse aux probabilités conditionnelles ayant la forme $P(x|E = e)$. Une première solution pour ce problème est d'échantillonner à partir de probabilité a posteriori $P(X|e)$. Cela se fait en utilisant le processus dit *rejection sampling* : on génère un échantillon x à partir de $P(X)$, et ensuite on rejette chaque échantillon qui n'est pas compatible avec e. L'échantillon retenu est alors généré à partir de la probabilité a posteriori $P(X|e)$.

D'autres méthodes similaires peuvent être appliquées pour ce genre de problèmes, citons *Likelihood Weighting* [FC90, SP90] et *Importance Sampling* [SP90].

Un autre groupe de méthodes stochastiques d'échantillonnage regroupe des méthodes basées sur les *Markov Chain Monte Carlo*. Ce groupe est divisé en deux types de méthodes : *Gibbs Sampling* et *Metropolis sampling* [Pea87, DC93] (Voir [KF09], Chapitre 12). Étant donnée une distribution de probabilité π, ces méthodes définient une chaîne de Markov dont la distribution stationnaire est π. Elles permettent ainsi de tirer aléatoirement un élément selon la loi π. L'échantillonnage de Gibbs est une adaptation particulière de l'échantillonnage de Metropolis-Hastings. Le principe de l'algorithme de Gibbs est décrit par l'algorithme 2. Il consiste à générer une séquence d'échantillons en commençant par générer un échantillon des variables non observées à partir d'une distribution initiale ; par exemple, le réseau peut être utilisé pour générer un échantillon en utilisant l'algorithme *forward*

Algorithme 2 Algorithme Échantillonneur de Gibbs

Entrées: $\mathbb{X}$ {*ensemble des variables à échantillonner*}
Φ {*ensemble des facteurs définissant* P_Φ}
$P^{(0)}(X)$ {*distribution initiale*}
N {*nombre d'échantillons*}
Sorties: $x^{(0)}, \ldots, x^N$ {*ensemble des échantillons*}

1: Générer $x^{(0)}$ à partir de $P^{(0)}(X)$,
2: **Pour** $t = 1, \ldots, N$ **Faire**
3: $x^{(t)} \leftarrow x^{(t-1)}$
4: **Pour** chaque $X_i \in \mathbb{X}$ **Faire**
5: Générer $x_i^{(t)}$ à partir $P_{\Phi(X_i|x-i)}$
{ *Changer* X_i *dans* $x^{(t)}$}
6: **Fin Pour**
7: **Fin Pour**

sampling. En partant de ce premier échantillon, l'algorithme de Gibbs itère, pour chaque variable non observée, en échantillonnant une nouvelle valeur pour chaque variable sachant ce premier échantillon pour toutes les autres variables.

Pour appliquer cet algorithme dans le cas où on a une évidence, on peut commencer par réduire tous les facteurs en utilisant cette observation e, ainsi la distribution P_Φ utilisée dans l'algorithme correspond à $P(\mathbb{X}|e)$.

Dans le cas où les variables sont continues, il est possible d'utiliser l'algorithme *Forward Sampling* (Algorithme 1) pour générer un ensemble d'échantillons selon la loi d'un RB gaussien ou hybride, du moment où nous possédons une méthode d'échantillonnage à partir d'une densité continue de chaque variable X_i sachant ses parents (la méthode *rejection sampling* par exemple). Toutefois, cet algorithme n'est pas applicable dans le cas de RB hybride en présence d'une évidence. Les méthodes *Importance sampling* et *likelihood Weighting* sont plus appropriées dans ce cas.

Parmi ces méthodes présentées pour le cas des modèles graphiques probabilistes dirigés, nous distinguons la méthode *Importance sampling* et les méthodes basées sur l'approche *Markov Chain Monte Carlo* qui sont applicables dans le cas de modèles graphiques non dirigés (Champs de Markov) et ceux partiellement dirigés (Chain graph) qui vont être présentés dans les sections suivantes.

2.6 Les UG : undirected graphs

2.6.1 Approche générale

Un graphe de dépendance non orienté est un graphe dont toutes les arêtes sont non dirigées. Un tel modèle graphique non dirigé est connu sous le nom de Champ de Markov (ou Markov random field), noté MRF. Il est défini par sa structure de graphe quelconque et non orientée. Deux variables connectées par un arc sont *voisins*. L'indépendance que représente un Champ de Markov est résumée par la propriété de Markov globale : ***une variable est indépendante de toutes les autres sachant ses voisins.***

Définition 6 (*Clique*)

Une clique *c est soit un singleton soit un ensemble de noeuds mutuellement voisins tel que pour toute paire $\{u, v\} \in c$ nous avons $v \in V_u$ (voisinage de u). Une clique est dite maximale si aucune clique ne la contient.*

Définition 7 (*Facteur*)

Soit V un ensemble de variables aléatoires. Un facteur *ψ est défini par une fonction sur $Val(V) \rightarrow \Re$.*

Conformément à l'équation 2.1, la distribution de probabilité d'un MRF est un produit de fonctions (facteurs) (dites potentiels de clique) définies chacune sur une clique du graphe. La factorisation d'une densité de probabilité d'un graphe non dirigé est donnée par :

$$f(x) = \prod_{c \in C} \psi_c(x)$$

où C est l'ensemble des cliques de G et $\psi_c(x)$ est une fonction non négative qui dépend de x_c uniquement. Les fonctions $\psi_c(x)$ représentent alors les fonctions élémentaires f_i de l'équation 2.1.

Un Champ de Markov est alors défini en deux étapes : premièrement, un graphe qui décrit la structure du modèle définissant implicitement les cliques, deuxièmement, la distribution de probabilité qui est exprimée sous la forme d'un produit de fonctions de variables dans chaque clique et quelques paramètres.

Le nombre de variables dans chaque clique intervient dans l'efficacité des calculs effectués par le modèle. Plus petit est le nombre de variables dans chaque clique, plus efficace est le modèle. La totalité des paramètres correspondant aux fonctions de facteurs forme l'ensemble des paramètres associé à la structure de graphe donnée.

Contrairement au cas des RB, les facteurs ne correspondent pas aux probabilités ou probabilités conditionnelles. Par conséquent, les paramètres ne sont pas intuitivement compréhensibles ce qui rend leur estimation à partir de données plus difficile.

2.6.2 Apprentissage des paramètres

L'apprentissage des paramètres d'un champ de Markov (dont le graphe est non dirigé) s'avère différent de celui des réseaux bayésiens. La raison la plus importante de ces différences revient probablement au fait que les champs de Markov sont caractérisés par une normalisation globale tandis que les réseaux bayésiens sont caractérisés par des normalisations locale au sein de chaque densité de probabilité conditionnelle. Ce facteur global englobe tous les paramètres du réseaux et empêche la décomposition du problème et l'estimation locale des paramètres, ce qui rend la tâche d'estimation des paramètres au sein d'un facteur global difficile de point de vue calcul. Il faut alors avoir recours aux méthodes itératives comme la descente du gradient pour optimiser les paramètres d'un champ de Markov.

Le problème de l'apprentissage des paramètres d'un champ de Markov est dû au fait qu'il faut faire de l'inférence sur tout le réseau à chaque itération, ce qui rend l'estimation des paramètres non faisable. L'estimation Bayésienne des paramètres, qui nécessite des intégrations sur l'espace des paramètres, est plus difficile du moment où il n'existe pas une forme pour l'expression des paramètres a posteriori.

Ces complexités ont conduit à l'utilisation de méthodes d'inférence approchée pour pouvoir calculer le gradient, ou encore à l'apparition d'approches qui ne cherchent plus à optimiser la vraisemblance mais une nouvelle fonction (*pseudo-vraisemblance*, *contrastive divergence*, *maximum-margin*).

Les méthodes utilisant l'inférence approchée, utilisent l'algorithme de *message locaux* pour la propagation de l'information dans le réseau et des approches d'échantillonnage.

La méthode qui essaye de maximiser une fonction de "pseudo-vraisemblance" a été proposée par [Bes77] et plusieurs extensions ont été proposées dans [HO02] et [MPDW06]. La cohérence asymptotique des fonctions de vraisemblance et de pseudo-vraisemblance est montrée dans [Gid88]. L'utilisation des méthodes d'estimation à base de marge *maximum-margin* pour les modèles probabilistes a été initialement proposée par [Col02]. [Hin02] et [TWO+03] ont introduit l'approche *contrastive divergence*.

2.6.3 Apprentissage de structure

L'apprentissage de structure des modèles non-dirigés à partir de données n'a pas été largement étudié comme le cas des modèles dirigés (RB). On peut utiliser, dans ce cas aussi, les méthodes à base de recherche des indépendances conditionnelles et à base de scores. A cause des difficultés découlant de la forme de la fonction de vraisemblance, le calcul des scores Bayésiens (où le calcul de score d'un modèle nécessite d'intégrer sur tous les paramètres) n'est pas faisable et même des approximations ne sont généralement pas pratiques. On utilise généralement une fonction de score plus simple, combinant un terme de vraisemblance et un terme de pénalité [KF09]. On cherche ensuite dans un certain espace de structures celles qui maximisent cette fonction de score.

Face à tous ces problèmes reliés essentiellement à l'inférence, certains travaux se sont orientés vers la recherche et l'apprentissage de réseaux de Markov de largeur bornée (bounded tree-whidh) qui est aussi un problème NP-difficile [Sre01] lorsque la largeur est supérieure à 1. Le travail de [CL68] a montré qu'il est possible de trouver l'arbre (dont largeur = 1) qui maximise la vraisemblance en un temps quadratique.

Un autre type d'approche est proposé par [FHT08]. Son principe consiste à utiliser la méthode *graphical lasso* pour estimer la matrice de covariance à partir de données supposées suivre une distribution gaussienne multivariée. Une valeur nulle dans l'inverse de cette matrice correspondra à une indépendance entre les variables correspondantes conditionnellement aux autres.

[KF09] contient une discussion plus détaillée pour les différentes méthodes qui ont été proposées pour l'apprentissage de structure d'un MRF.

2.7 Les Chain Graphs : Partially Directed Acyclic Graph

Les chain graphs, qui admettent les deux types d'arcs dirigés et non-dirigés, mais pas de cycles partiellement dirigés, ont été introduits comme une généralisation naturelle des DAG et des graphes non dirigés [LW89b]. Les premières motivations pour les chain graphs sont reliés au fait que l'introduction d'arcs non dirigés permet de modéliser des *interventions simultanées* [Fry90] ou des *associations symétriques* [LW89b].

Une structure d'un PDAG (*Partially Directed Acyclic Graph*) K peut définir

une factorisation de la distribution de probabilité de K selon la propriété de Markov pour les chains graphs. Intuitivement, la factorisation des PDAG représente la distribution sous la forme d'un produit de chaque chaine composant le graphe sachant ses parents. Une telle représentation est appelée *chain graph model*.

Chaque chaine composante K_i dans le graphe est associé à une CRF (*Conditional Random Field*) qui défini $P(K_i|Pa_{K_i})$, la distribution conditionnelle de K_i sachant ses parents dans le graphe. La distribution $P(K_i|Pa_{K_i})$ est définie par l'utilisation de facteurs associés à K_i pour définir un CRF dont les variables cibles sont K_i et les variables observées sont Pa_{K_i}.

soit K un PDAG, et $K_1, \ldots, K_l$ ses chaines composantes. Une distribution d'un chain graph est définie via l'ensemble des facteurs $\phi_i(C_i)(i = 1, \ldots, m)$ où chaque C_i est un sous-graphe complet du graphe moralisé $M[K]$. Chaque facteur $\phi_i(C_i)$ est associé à une chaine composante K_i, où $C_i \subset K_i \cup Pa_{K_i}$ et défini $P(K_i|Pa_{K_i})$ comme un CRF avec ces facteurs.

La factorisation d'une distribution d'un chain graph est alors donnée par :

$$P(X) = \prod_{i=1}^{l} P(K_i|Pa_{K_i})$$

Nous ne détaillons pas ce cas de modèles graphiques partiellement dirigés. Une étude détaillée de la propriété de Markov pour les chain graph peut être trouvée dans [Fry90]. [LR01] étudie les chain graphs comme généralisation des DAG et des graphes non dirigés, [Drt09] a considéré le cas discret des chain graphs et a étudié l'apprentissage des paramètres par MV dans ce cadre.

2.8 Cas particulier des structures arborescentes

Dans la théorie des graphes, un polyarbre est un graphe dirigé avec au maximum un seul chemin dirigé entre chaque paire de variables. Autrement dit, un polyarbre est un DAG qui ne contient pas de cycles non dirigés.

Un modèle de polyarbre pour une densité de variable X est défini par :

- une structure de DAG P dont le squelette est sans circuit et connecté,
- un ensemble de noeuds qui représentent $X = X_1, ..., X_n$,
- un ensemble de densités conditionnelles $P_P(Xi|Pa_P(Xi))$, où $Pa_P(X_i)$ est l'ensemble des variables en bijection avec les parents de X_i dans P.

La structure P représente graphiquement la factorisation de densité

$$P_P(X) = \prod_{i=1}^{n} P_P(Xi|Pa_P(Xi))$$

Les paramètres du modèle sont alors spécifiés par le vecteur de distributions conditionnelles :

$$\theta_P = (P_P(Xi|Pa_P(Xi)))_{i=1}^{n}$$

Nous notons que conformément avec cette définition, un polyarbre contient $n-1$ arcs.

On peut définir des sous-classes imbriquée P^p des structures de polyarbres en imposant des contraintes sur le nombre maximum p de parents par chaque noeud.

La sous-classe la plus petite des structures polyarbres est la classe des arbres où les noeuds ont exactement un seul parent ($p = 1$). Les modèles d'arbres ont la propriété essentielle de ne pas contenir de $v - structures$ (voir annexe B). Donc chaque arbre dirigé est un polyarbre, mais chaque polyarbre n'est pas essentiellement un arbre. ***Le modèle de dépendances d'un arbre peut être "lu" sans tenir compte de la direction de ses arcs.***

2.8.1 Inférence dans les arbres

Comme il a été mentionné dans la section 2.3, le problème de l'inférence est NP-difficile mais pour les structures simples (polyarbres), l'inférence peut être effectuée efficacement en utilisant l'algorithme proposé par [Pea85] appelé *Message locaux*. ***Cet algorithme est aussi applicable dans le cas des arbres de Markov (arbres non dirigés), en plus des arbres dirigés et des polyarbres.***

[Pea88] a adapté l'algorithme de *Message locaux* pour le cas où les variables sont continues. Le principe est resté le même, sauf que les quantités échangées entre un noeud et ses voisins (parents ou fils), qui étaient des vecteurs de dimension en relation avec la variable et l'ensemble de ses parents, sont adaptés pour être caractérisées par les moyennes et les variances des densités conditionnelles gaussiennes correspondantes.

Complexité de l'inférence Dans le cas où notre modèle est représenté par un arbre (dirigé ou non), une borne supérieure pour le nombre de valeurs stockées

dans l'arbre est donnée par [Nea04] :

$$n(k^2 + 2k) + 2(n-1)k \propto O(nk^2)$$

où
n = le nombre de noeuds dans l'arbre
k = le nombre maximum de valeurs d'un noeud

Le nombre de multiplications nécessaires pour calculer toutes les probabilités conditionnelles est donnée par :

$$n(2k^2 + 2k + kc) \propto O(nk^2 + nkc)$$

où
c = le nombre maximum de fils dans tous les noeuds

Pour les polyarbres, si la structure du graphe est éparse (i.e., chaque noeud n'a pas beaucoup de parents), l'algorithme reste efficace en terme d'espace mémoire et temps de calcul. Cependant, si un noeud possède plusieurs parents, la complexité de l'espace de stockage seule devient intraitable.

2.8.2 Apprentissage de structure des arbres

[CL68] a montré qu'une distribution jointe arbitraire peut être approchée de façon optimale par une distribution d'arbre, en utilisant les probabilités marginales des paires de variables. [CL68] a proposé un algorithme pour apprendre l'arbre de Markov (non dirigé) optimal à partir de données en utilisant l'algorithme MWST (Maximum Weight Spanning Tree). Le principe de cette méthode est décrit dans l'algorithme 3.

Cette méthode s'applique à la recherche de structure d'un réseau bayésien en fixant un poids à chaque arête potentielle $A - B$ de l'arbre. Ce poids peut être par exemple l'information mutuelle entre les variables A et B comme proposé dans [CL68] (voir formule 2.7), ou encore la variation du score local (décomposable) lorsqu'on choisit B comme parent de A comme proposé dans [HGC95] (voir formule 2.8).

$$W_{CL}(X_A, X_B) = \sum_{x_a, x_b} P(X_A = x_a, X_B = x_b) \log \frac{P(X_A = x_a, X_B = x_b)}{P(X_A = x_a) P(X_B = x_b)}$$

Algorithme 3 Algorithme de MWST ([CL68])

Entrées: $D,\ n$
Sorties: T

1: $MI = [0]_{n\times n}$,
2: **Pour** $i = 1, \ldots, n$ **Faire**
3: **Pour** $j = i+1, \ldots, n$ **Faire**
4: $MI\,[i,j] = ComputeMI(X_i, X_j, D)$
5: $MI\,[j,i] = MI\,[i,j]$
6: **Fin Pour**
7: **Fin Pour**
8: $T = Kruskal(MI)$

$$= \sum_{a,b} \frac{N_{ab}}{N} \log \frac{N_{ab}N}{N_{a.}N_{.b}} \tag{2.7}$$

$$W\,(X_A, X_B) = score\,(X_A, X_B) - score\,(X_A, \oslash) \tag{2.8}$$

où $score\,(X_A, X_B)$ est le score local en X_A en supposant qu'il a X_B comme parent, et $score\,(X_A, \oslash)$ est le score local en X_A en supposant qu'il n'a aucun parent.

A partir de ce calcul de score, nous calculons une matrice MI symétrique tel que : $MI_{ij} = MI_{ji} = W(X_i, X_j)$, avec $i \neq j$.

Une fois cette matrice de poids définie, il suffit d'utiliser un des algorithmes standards de résolution du problème de l'arbre de poids maximal comme la méthode de *Kruskal* [Kru56] définie par l'algorithme 4. L'arbre de poids maximal est un arbre non dirigé reliant toutes les variables, donc il doit par la suite être dirigé en choisissant une racine puis en parcourant l'arbre par une recherche en profondeur jusqu'aux noeuds feuilles. La racine peut être choisie soit aléatoirement, soit à l'aide de connaissance a priori.

De point de vue complexité, la recherche de l'arbre optimal est quadratique en fonction du nombre de variables du domaine. En effet, la matrice d'information mutuelle W est de taille n^2, avec n est le nombre de variables. On aura donc à calculer $n(n-1)/2$ informations mutuelles. L'algorithme de *Kruskal* est de complexité $E\ log(E)$, où E est le nombre d'arcs considérés lors de la recherche. La complexité de l'algorithme de Chow et Liu est alors $E\ \log(E) \propto O(n^2 log(n))$ où n est le nombre de variables

Algorithme 4 Algorithme de Kruskal ([Kru56])

Entrées: $MI_{n\times n}$
Sorties: T

1: Classer les arêtes par ordre de mesure décroissante $(u_1, u_2, u_3, \ldots, u_l)$,
2: $T = \{\}$, $i = 1$,
3: **Tantque** $|T| < n - 1$ **Faire**
4: **Si** $T \cup \{u_i\}$ est sans cycle **Alors**
5: $T \leftarrow T \cup \{u_i\}$
6: **Sinon**
7: $i \leftarrow i + 1$
8: **Fin Si**
9: **Fin Tantque**

La recherche du polyarbre optimal qui représente une distribution est un problème plus difficile. Il est possible de construire un polyarbre optimal si les données suivent exactement une telle structure, mais il n'est pas connu comment trouver le meilleur polyarbre approchant les données si ces dernier ne possèdent pas une représentation par polyarbre. [RP88] décrit une extension de l'algorithme de Chow et Liu pour rechercher le polyarbre optimal à partir de données. Ce principe est décrit par l'algorithme 5.

Cet algorithme de recherche du polyarbre optimal est de complexité polynomiale en fonction du nombre de variables.

[OOM04] propose un autre algorithme de recherche du polyarbre optimal à partir de données. Cet algorithme utilise aussi l'algorithme de Chow et Liu pour rechercher une structure non orientée, ensuite utilise des tests d'indépendances et un algorithme de parcours en profondeur pour orienter le polyarbre.

2.9 Problèmes des espaces de grande dimension

Nous avons évoqué dans la section 1.5 que dans les espaces de grande dimension, les données disponibles sont souvent de taille très petite en comparaison avec le nombre de variables.

Lorsque les données disponibles sont de taille réduite, l'augmentation du nombre de variables entraine, dans le cas des modèles paramétriques, une augmentation du nombre de paramètres nécessaires pour définir un modèle et l'apparition du phénomène du sur-apprentissage (déjà discuté dans la section 1.5). De plus, l'augmentation de la dimension de l'espace des paramètres conduit à une augmen-

Algorithme 5 Algorithme de recherche du polyarbre optimal ([RP88])

Entrées: D, n
Sorties: PT

1: $PT = MWST(D, n)$,
2: **Répéter**
3: Recherche dans les noeuds internes de PT les noeuds qui ont plus qu'un parent (V-structures)
4: **Si** un noeud dans PT ayant plusieurs parents est détecté **Alors**
5: déterminer la direction de toutes ses branches en utilisant le test de V-structures,
6: **Fin Si**
7: **Pour** chaque noeud dans PT qui a au moins un arc rentrant **Faire**
8: résoudre la direction de toutes ses branches adjacentes en utilisant des tests statistiques d'indépendance conditionnelle
9: **Fin Pour**
10: **Jusqu'à** aucune direction supplémentaire ne peut être découverte
11: **Si** il reste des branches non orientées dans PT **Alors**
12: étiqueter les *"non déterminé"* et avoir recours aux sémantiques externes pour compléter l'orientation.
13: **Fin Si**

tation exponentielle des demandes en temps de calcul pour trouver un ensemble de paramètres optimaux.

La factorisation en densités jointes que les modèles graphiques fournissent permet de réduire la complexité du problème. En effet, les modèles graphiques profitent des indépendances entre les variables pour diminuer le nombre de paramètres nécessaires pour représenter le problème et ensuite réduire les demandes en temps de calcul et espace mémoire pour, à la fois, l'apprentissage des paramètres et l'inférence.

Toutefois, l'augmentation de la taille du problème (nombre de variables) rend l'utilisation des modèles graphique probabilistes difficile à mettre en oeuvre. En effet, nous avons vu dans la section 2.3 que l'inférence exacte est un problème NP-difficile lorsque la structure d'un modèle graphique est quelconque. L'augmentation de la dimension du problème aggrave de plus en plus ce problème et rend impossible d'effectuer de tels calculs. Toutefois, il est possible d'effectuer les algorithmes d'inférence efficacement si la structure du modèle n'est pas trop complexe, et encore plus efficacement si elle est simple (structure arborescente).

D'autre part, les algorithmes d'apprentissage de structure ne sont pas efficaces du moment où la taille du problème dépasse quelques dizaines de variables. Cela

est dû essentiellement à la taille super-exponentielle de l'espace de recherche. Les méthodes de recherche locales restent le seul refuge pour un tel apprentissage.

De plus, l'apprentissage d'un modèle à partir de peu de données peut souffrir du phénomène du sur-apprentissage. Pour cela, les données sont souvent apprises par un modèle simple avec un petit nombre d'arcs [WJ08].

L'utilisation de modèles simples peut ainsi tirer profit de la simplicité et la scalabilité de l'inférence et de l'apprentissage (des paramètres et de structure) dans ce type de modèles.

2.10 Conclusion

Dans ce chapitre, nous avons présenté les modèles graphiques probabilistes, en commençant par dévoiler l'apport de ces outils dans le cadre de l'estimation de la densité. En effet, la factorisation de la loi jointe de ces MGP permet de réduire la complexité du problème traité en diminuant le nombre de paramètres considérés.

Nous avons aussi discuté les différents types des MGP, notamment les DAG, les Champs de Markov et les Chain graph. Pour chacun de ces types, nous avons étudié les différents algorithmes d'inférence ainsi que les algorithmes d'apprentissage de paramètres et de structure à partir de données.

Ensuite, un algorithme générique de simulation de nouvelles données selon la loi d'un MGP a été présenté ainsi que d'autres algorithmes couramment utilisés.

Dans la section 2.8 nous avons considéré un cas particulier des MGP où la structure non orientée du graphe est un arbre et nous avons discuté les différents algorithmes d'inférence et d'apprentissage de point de vue complexité et passage à l'échelle aux domaines de grande dimension.

La dernière section 2.9 a servi à discuter des problèmes des espaces de grande dimension et l'apport des modèles simples (arbres) dans ce cadre. Ces modèles simples feront le sujet du prochain chapitre où nous présenterons les mélanges de modèles graphiques probabilistes simples pour l'estimation de la densité.

Chapitre 3

Les mélanges de modèles pour l'estimation de densité

3.1 Introduction

Les mélanges de distributions ont fourni un ensemble d'approches mathématiques pour la modélisation statistique d'une grande variété de phénomènes aléatoires. Vue leur utilité comme méthodes de modélisation flexibles, les modèles de mélange ont continué à avoir un succès et un intérêt croissant du point de vue pratique et théorique.

Les domaines dans lesquels les modèles de mélange ont été appliqués avec succès comprennent l'astronomie, la biologie, la génétique, la médecine, la psychiatrie, l'économie, l'ingénierie et le marketing.

Parmi les premières analyses impliquant les mélanges de modèles, citons le travail du biostatisticien Karl Pearson qui date de plus de 100 ans. Il a utilisé dans [Pea94] un mélange de deux fonctions de densité de probabilité normales de moyennes et variances différentes.

Dans ce chapitre, nous introduisons dans un premier temps le principe général des mélanges de modèles ainsi que le cadre général de cette approche *le Bayesian Model Averaging* où la taille du mélange est infinie. Dans la section 3.3, nous présentons les approches de mélanges de taille finie dont l'apprentissage à partir de données d'observation est basé sur l'algorithme EM ainsi que quelques cas particuliers pour certains types de modèles élémentaires. Nous passons ensuite dans la section 3.4 à un autre type d'approches basées sur le principe du *Perturb*

et Combine pour l'apprentissage de mélanges de modèles finis.

Enfin, la section 3.6 est consacrée à une discussion de ces différentes approches dans un cadre spécifique où le nombre d'individus dans la base d'apprentissage est trop petit devant la grande dimension du problème.

3.2 Principe général

Soit $(x^1, ..., x^N)$ un ensemble de données d'observations jointes de taille N. Nous définissons un mélange de distributions (modèles) (en général) comme étant une distribution de la forme

$$f(x) = \sum_{k=1}^{m} \lambda_k f^k(x)$$

avec

$$\lambda_k \geq 0, k = 1, \ldots, m; \qquad \sum_{k=1}^{m} \lambda_k = 1.$$

Les modèles f^k représentent les *composantes du mélange* et les poids λ_k sont appelés les *coefficients du mélange*. Les m modèles peuvent avoir des structures différentes et des paramètres différents.

Le mélange de modèles paramétriques

Dans le cas où les densités f_k sont des fonctions paramétriques, le mélange est dit aussi paramétrique et il est donné par :

$$f(x|\Psi) = \sum_{k=1}^{m} \lambda_k f^k(x, \theta_k)$$

avec θ_k est l'ensemble des paramètres de la $k$ème composante f_k du mélange. L'apprentissage d'un mélange de modèles pour l'estimation de la densité de probabilité paramétrique revient alors à déterminer $\Psi = \{\theta_1, \ldots, \theta_m, \lambda_1, \ldots, \lambda_m\}$, l'ensemble de tous les paramètres du mélange.

Le Bayesian model averaging

Soit $X = (X_1, ..., X_n)$ un ensemble de variables aléatoires discrètes, et $D = (x^1, ..., x^N)$ un ensemble de données d'observations jointes $x^i = (x_1^i, ..., x_n^i)$ générées indépendamment par une densité $P(X)$.

Dans toute l'approche Bayésienne, on suppose que $P(X)$ appartient à un espace de densités $\mathcal{D}$ décrit par un modèle $M \in \mathcal{M}$ et des paramètres $\theta_M \in \Theta_M$, et on déduit de l'ensemble de données un mélange de modèles décrit par l'équation suivante :

$$P_{\mathcal{D}}(X|D) = \sum_{M \in \mathcal{M}} P(M|D)\ P(X|M, D), \tag{3.1}$$

où $P(M|D)$ est la probabilité *a posteriori* sur l'espace des modèles $\mathcal{M}$ conditionnellement aux données D, et où $P(X|M, D)$ est l'intégrale :

$$P(X|M, D) = \int_{\Theta_M} P(X|\theta_M, M)\ dP(\theta_M|M, D). \tag{3.2}$$

$P_{\mathcal{D}}(X|D)$ est calculée par :

$$P_{\mathcal{D}}(X|D) = \sum_{M \in \mathcal{M}} P(M|D) \int_{\Theta_M} P(X|\theta_M, M)\ dP(\theta_M|M, D), \tag{3.3}$$

où $P(\theta_M|M, D)$ est la densité *a posteriori* des paramètres et $P(X|\theta_M, M)$ est la vraisemblance de l'observation X pour le modèle M avec les paramètres θ_M.

Notons que dans cette équation, l'estimation de la densité de probabilité de X ne se fait pas par l'apprentissage d'un seul modèle à partir des données mais par une *somme pondérée* sur *tous les modèles possibles*, pour *tous les paramètres possibles*. Toutefois, dans la pratique, il est empiriquement peu intéressant de faire un tel calcul.

Plusieurs travaux ont opté pour un mélange de taille finie obtenu selon différentes politiques : nous citons les mélanges de modèles de taille finie basés sur l'algorithme EM pour l'apprentissage de tous les paramètres du mélange, les méthodes qui se basent sur des techniques de perturbation soit de l'ensemble de données d'apprentissage (méthodes d'agrégation de modèles simples tels que le Bagging (Bootstrap aggregation) et le Boosting), soit de la procédure de génération des composantes du mélange (méthodes aléatoires par exemple).

Nous détaillons dans la suite ces deux politiques de construction des mélanges de modèles ainsi que les différentes techniques utilisées.

3.3 Les mélanges de modèles finis

3.3.1 Approche générale

L'estimation des paramètres du modèle de mélange paramétrique (qu'il soit gaussien ou non) a fait l'objet de nombreux travaux depuis l'introduction du modèle de mélange par [Pea94] et la méthode la plus fréquemment utilisée aujourd'hui est celle du *maximum de vraisemblance* (MV).

Le problème de l'estimation des paramètres d'un modèle de mélange peut être vu comme un problème paramétrique. Toutefois, le vecteur de paramètres Ψ contient l'ensemble des poids λ_k qui n'est pas représenté dans les données. Les équations de vraisemblance ainsi obtenues ne possèdent pas de solutions analytiques. [MP00] représente le problème sous la forme à données incomplètes, qui peut être résolu par l'algorithme EM, comme suit.

Chaque x_i dans l'ensemble des données $x = (x_1, \ldots, x_N)$ est considéré comme ayant résulté de l'une des composantes du modèle de mélange étant appris. z_i est un vecteur de dimension m avec $z_{ij} = (z_j)_i = 1$ *ou* 0, selon que x_j doit ou ne doit pas résulter de la i ème composante du mélange ($i = 1, \ldots, m; j = 1, \ldots, N$). Comme les z_i ne sont pas disponibles dans les données d'apprentissage, alors ces données sont vues comme incomplètes.

L'algorithme EM est utilisé conjointement avec le principe du MV pour estimer les paramètres Ψ du mélange.

La fonction log de vraisemblance $log(L_c(\Psi))$ pour les données complétées est alors données par :

$$logL_c(\Psi) = \sum_{i=1}^{m} \sum_{j=1}^{N} z_{ij} \{\log \lambda_i + \log f_i(x_i; \theta_i)\}$$

Algorithme EM et mélanges finis de modèles

L'algorithme EM [DLR77] est couramment utilisé conjointement avec le principe du MV pour estimer les paramètres Ψ du mélange. Il est appliqué à ce problème en traitant les z_{ij} comme données manquantes, et procède itérativement en deux étapes, E (pour Expectation) et M (pour Maximization). Son principe est le suivant :

Soit Ψ^0 est la valeur initiale de Ψ.

L'étape E : Cette étape consiste à calculer, à l'itération q, l'espérance de Z_{ij} (la variable aléatoire associée à z_{ij}) sachant la valeur du paramètre estimé à l'étape précédente et que $x = x_j$.

$$t_{ij}(x_i; \Psi^{(q)}) = E_{\Psi^{(q)}}(Z_{ij}|x) = P_{\Psi^{(q)}}(Z_{ij} = 1|x)$$

puisque Z_{ij} est à valeurs dans $\{0, 1\}$. La formule de Bayes permet finalement de formuler les probabilités t_{ij} que x_j doit résulter de f_i, et ce conditionnellement au paramètre courant $\Psi^{(q)}$, de la façon suivante :

$$\begin{aligned} t_{ij}^{q+1}(x_i; \Psi^{(q)}) &= \lambda_i^q f_i(x_i; \theta_i^{(k)}) / f(x_i; \Psi^{(q)}) \\ &= \lambda_i^q f_i(x_j; \theta_i^{(k)}) / \sum_{h=1}^{m} \lambda_h^q f_h(x_j; \theta_h^{(k)}) \end{aligned}$$

L'étape M : Cette étape consiste quant à elle à maximiser, à chaque étape q, l'espérance de la vraisemblance complète conditionnellement aux $t_{ij}^{(q)}$. Les proportions $\{\hat{\lambda}_i^q\}_{i=1}^m$ du mélange sont obtenues simplement par la relation :

$$\hat{\lambda}_i^q = \sum_{j=1}^{n} t_{ij}^{(q)} / N \quad (i = 1, \ldots, m)$$

Les estimateurs des paramètres $\{\theta_1, ..., \theta_n\}$ sont ainsi obtenus en résolvant les équations de vraisemblance correspondantes au modèle de mélange retenu.

Limitations de l'algorithme EM

L'algorithme EM découle naturellement des équations de vraisemblance et possède de bonnes propriétés statistiques en plus de sa simplicité de mise en oeuvre et de son efficacité. Cet algorithme assure une convergence vers un maximum local de la vraisemblance sous certaines conditions de régularité [Wu83].

Cependant, cet algorithme possède un certain nombre de limitations. D'abord, la valeur de l'estimateur à la convergence peut être fortement dépendante de son initialisation. Une autre limitation de l'algorithme EM est que la convergence peut être lente et l'algorithme peut même se trouver bloqué dans un point selle de la vraisemblance.

Ces limitations ont donné lieu à des développements complémentaires. Pour

pallier la limitation de la dépendance de la solution vis à vis de l'initialisation, il est courant dans la pratique de lancer plusieurs EM pour quelques itérations depuis des initialisations aléatoires et de choisir la valeur des paramètres associée à la plus grande vraisemblance comme initialisation d'un autre EM qui lui itèrera jusqu'à convergence. Parmi les variantes existantes de cet algorithme nous citons l'algorithme GEM (Generalized EM), proposé par [DLR77], qui permet de simplifier le problème de l'étape maximisation, l'algorithme CEM (Classification EM), proposé par [CG92], permettant de prendre en compte l'aspect classification lors de l'estimation, ainsi que l'algorithme SEM (Stochastic EM), proposé par [CD85], dont l'objectif est de réduire le risque de tomber dans un optimum local de vraisemblance. [KK09] a proposé une stratégie pour obtenir une bonne initialisation (ensemble d'arbres) de l'algorithme EM.

Choix de la taille du mélange

Dans les approches où l'apprentissage des paramètres du mélange est basé sur l'algorithme EM, la taille m du mélange (nombre de composantes) doit être connue préalablement et ne peut pas être estimée par MV. Le choix de cette valeur est un problème important et difficile. Ce paramètre m d'un mélange F_m est couramment estimé en utilisant la théorie générale de la sélection de modèles. Ce paramètre est estimé de façon à maximiser la quantité

$$\log(F_m) - a_{mn}$$

avec a_{mn} est un terme de pénalisation qui satisfait $a_{m+1,n} \geq a_{mn}$ qui décourage la sélection d'un modèle avec un nombre plus grand de composantes.

Dans le cadre de l'estimation de la densité, les critères les plus couramment utilisés AIC [Aka70] et BIC [Sch78] paraissent adéquats pour choisir le nombre m de composantes d'un mélange. Le terme de pénalisation a_{mn} s'exprime selon le choix du critère considéré ([Ler92]). Pour le critère AIC $a_{mn} = dim(F_m)$, et pour le critère BIC $a_{mn} = (1/2)(\log n)dim(F_m)$; où $dim(F_m) = m - 1 + m\ dim(\theta)$.

Plusieurs travaux se sont intéressés à trouver la meilleure valeur de m dans le cadre de l'estimation de la densité et la classification automatique. Parmi ces travaux nous citons [Ler92] qui a montré que sous certaines conditions, les critères de pénalisation de la fonction *log-vraisemblance* (AIC et BIC) ne sous estiment pas le vrai nombre de composantes d'un mélange de normales. Citons aussi [RW97] et [BCG03] qui ont discuté l'utilisation des critères AIC et BIC dans ce cadre

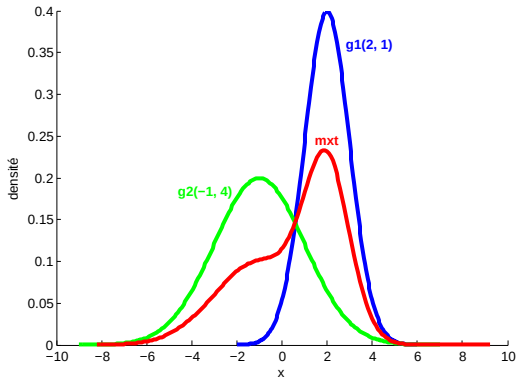


Figure 3.1. Mélange de deux gaussiennes (bleue et verte) avec des proportions égales (50%). La densité du mélange est représentée en rouge.

d'estimation de densité.

Estimation par maximum a posteriori

L'algorithme EM peut être simplement modifié pour produire une estimation bayésienne (maximum a posteriori) correspondant à une densité a priori $p(\Psi)$ de Ψ. L'étape E est effectivement la même que celle correspondant à la recherche du maximum de vraisemblance de Ψ. L'étape M est différente puisque l'on maximise $p(\Psi|x)$ au lieu de $p(x|\lambda_i^q)$. La fonction de recherche de $\Psi^{(q)}$ dans le processus de maximisation est augmentée par le log de la densité a priori $\log p(\Psi)$. [MP00] détaille d'avantage l'approche bayésienne pour les mélanges de modèles dans son chapitre 4.

Ce principe général des mélanges finis de modèles, dont l'apprentissage à partir de données d'observations se base sur l'algorithme itératif EM, a été appliqué dans différents domaines et sur différents types de modèles élémentaires tels que les distributions normales (section 3.3.2) et les modèles graphiques probabilistes (section 3.3.3). Nous détaillons brièvement dans ce qui suit certains exemples.

3.3.2 Mélanges de gaussiennes

Un modèle de mélange gaussien est une estimation de la distribution de variables aléatoires en utilisant la somme de plusieurs noyaux gaussiens dont la variance, la moyenne et l'amplitude sont variables. Dans ce cas, les fonctions f_k ont la forme d'une gaussienne.

L'algorithme EM est utilisé pour estimer les paramètres du mélange gaussien et l'étape M revient alors à calculer, à l'étape q, les estimateurs des paramètres

d'une distribution gaussienne multivariée (voir Eq. 1.7) et les poids du mélange λ_i comme suit :

$$\hat{\lambda}_i^q = \frac{N_i^{(q)}}{N}, \qquad N_i^{(q)} = \sum_{j=1}^{N} t_{ij}^{(q)}$$

$$\hat{\mu}_i^q = \frac{1}{N_i^{(q)}} \sum_{j=1}^{N} t_{ij}^{(q)} x_j,$$

$$\hat{\Sigma}_i = \frac{1}{N_i^{(q)}} \sum_{j=1}^{N} t_{ij}^{(q)} (x_j - \mu_i^q)(x_j - \mu_i^q)^T$$

3.3.3 Mélange de modèles graphiques probabilistes

Plusieurs travaux ont considéré que les fonctions f_k composantes du mélange peuvent être des modèles graphiques probabilistes. Nous citons [TMC+97] qui a utilisé un mélange de DAG appris en utilisant l'algorithme EM. Ce travail a considéré des problèmes dont les variables sont continues et ayant la forme d'une distribution normale. [TMCH97] a aussi considéré les variables continues et a utilisé un mélange de modèles graphiques gaussiens dont l'apprentissage est basé sur l'algorithme EM.

[CS96] a utilisé un mélange de distributions factorielles sous la forme d'un modèle de Bayes naïf (où une variable est choisie pour être la variable *Classe* et ensuite utilisée pour des problèmes de classification). [KMT96] a aussi utilisé un mélange de distributions factorielles mais avec une variable *Classe* cachée pour la classification. [LD05] a proposé aussi d'utiliser un modèle de Bayes naïf avec une variable *Classe* cachée pour l'estimation de probabilité.

Vu que l'apprentissage des modèles graphiques dont la structure est quelconque est un problème NP-difficile [CGH94], certains travaux se sont restreint à l'utilisation des modèles à structure simple (arbres, polyarbres...) pour tirer profit de leur simple manipulation (apprentissage des paramètres et inférence). Citons [MP99] qui a considéré des mélanges formés par un ensemble de modèles graphiques probabilistes simples dont la structure est un arbre et les variables sont discrètes. [Kir08, KK09] ont aussi considéré les mélanges d'arbres appris par l'algorithme EM pour l'estimation de densité multivariée. [LZLT09] a proposé un algorithme à base de mélanges d'arbres dans le cas de variables discrètes et continues.

Nous détaillons brièvement dans ce qui suit deux algorithmes parmi ceux cités : **NBE** proposé par [LD05] et *MixTree* proposé par [MP99].

3.3.3.1 Mélange de modèles de Bayes naïfs (NBE)

[LD05] propose une méthode d'apprentissage de modèles de Bayes naïfs pour l'estimation de la probabilité. La propriété "naïve" de ces modèles revient à l'indépendance de toutes ses variables $X_1, \ldots, X_n$ sachant la variable classe C. La distribution de probabilité de ce modèle peut alors s'écrire

$$P(X_1, X_2, \ldots, X_n, C) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

La distribution marginale d'une variable X peut être calculée en sommant sur toutes les valeurs de C :

$$P(X) = \sum_{k=1}^{m} P(C = k) \prod_{i=1}^{n} P(X_i|C = k) \tag{3.4}$$

Avec m la cardinalité de la variable classe C. Ce modèle de Bayes naïf peut être vu comme un mélange de graphes vides où la dimension de la variable *Classe* correspond à la taille du mélange (nombre de composantes du mélange), et $\{P(C = k)\}_{k=1,\ldots,m}$ les coefficients du mélange. En effet, l'équation 3.4 peut s'écrire comme suit :

$$P(X) = \sum_{k=1}^{m} \lambda_k P_k^v(X)$$

avec $\lambda_k = P(C = k)$ représente la probabilité que la variable classe $C = k$, et $P_k^v(X) = \prod_{i=1}^{n} P_k^v(X_i)$ est la loi jointe du $k$iéme graphe vide.

Dans [LD05], l'apprentissage des paramètres de ce modèle de Bayes naïf est basé sur l'algorithme EM. Le principe de cette méthode est décrit par l'algorithme 6. Cet algorithme comprend deux phases essentielles : extension (ajout de nouvelles composantes au mélange) et raffinement (apprentissage jusqu'à convergence). L'étape de raffinement utilise l'algorithme EM pour apprendre les paramètres des composantes et comprend une étape d'élagage qui sert à enlever les composantes de poids faible du mélange afin d'accélérer la recherche.

[LD05] a montré que les mélanges de graphes vides sont des approximateurs universels.

3.3.3.2 Mélange d'arbres (MixTree)

Ce mélange d'arbres (proposé par [MP99]) est utilisé pour l'estimation de densité de probabilité. Ce travail a montré que les mélanges d'arbres ont presque

Algorithme 6 Algorithme NBE ([LD05])

Entrées: D {*ensemble de données*}

Répéter
  ajouter k éléments au mélange
  Répéter
    E step : attribuer des exemples pour les éléments du mélange
    M step : ré-estimer les paramètres du modèle
    chaque 5 itérations, élaguer les éléments ayant des poids faibles
  Jusqu'à convergence sur l'ensemble de données de validation
  $k = 2 \times k$
Jusqu'à convergence sur l'ensemble de données de validation
exécuter les étapes **E** et **M** 3 fois en utilisant tous les données d'apprentissage

les mêmes avantages que les arbres de point de vue temps de calcul. Comme les mélanges de graphes vides, les mélanges d'arbres sont des approximateurs universels dans l'espace des distributions. Un mélange d'arbres est défini comme une distribution de la forme :

$$Q(X) = \sum_{k=1}^{m} \lambda_k \, T^k(X) \tag{3.5}$$

avec $\lambda_k \geq 0, \ k = 1, \ldots, m; \quad \sum_{k=1}^{m} \lambda_k = 1.$

Les distributions d'arbres T^k représentent les composantes du mélange et λ_k sont les coefficients de pondération.

Le principe de la méthode MixTree est décrit par l'algorithme 7. L'apprentissage des paramètres du mélange d'arbres se fait par une procédure basée sur le principe de l'algorithme EM décrit dans la section 3.3.1. La structure des arbres éléments du mélange se fait par le moyen de l'algorithme MWST (Algorithme 3).

La complexité en temps de calcul de cette méthode par itération de l'algorithme EM est quadratique $O(n^2)$ en fonction du nombre de variables. La complexité en espace mémoire de cette méthode est aussi quadratique en fonction du nombre de variables.

3.4 Mélanges de modèles par échantillonnage

Dans la section précédente, nous avons présenté différents mélanges de modèles finis utilisant l'algorithme EM pour l'apprentissage des paramètres. Ces approches utilisent des techniques spécifiques à chacun des domaines pour construire le mé-

Algorithme 7 Algorithme MixTree ([MP99])

Entrées: $D = x^1, ..., x^N$ {*ensemble de données*}
$\mathcal{M} = \{m,^k, \lambda^k, k = 1, .., m\}$ {*modèle initial*}

1: **Répéter**
2: **E step :** pour $k = 1, ..., m$, $i = 1, ..., N$
3: calculer γ_k^i {*les probabilités a posteriori de chaque arbre ayant généré une donnée* x^i}
4: calculer $P^k(X^i)$ {*distribution de probabilité sur les données* }
5: **M step :** pour $k = 1, ..., m$
6: calculer les poids λ_k
7: calculer les marginales P_v^k, P_{uv}^k, $u, v \in V$
8: calculer les informations mutuelles I_{uv}^k, $u, v \in V$
9: $E_{T^k} \leftarrow ComputeMWST(\{I_{uv}^k\})$
10: $T_{uv}^k \leftarrow P_{uv}^k$, $T_v^k \leftarrow P_v^k$ pour $(u, v) \in E_{T^k}$
11: **Jusqu'à** convergence

lange de modèles. Nous présentons dans cette section d'autres approches dont le principe de construction du mélange est basé sur l'échantillonnage dans l'espace des modèles et le principe du *Perturb et Combine*.

3.4.1 Cadre général : Principe du Perturb and combine

Un modèle (de classification ou de régression) est dit instable si une perturbation mineure dans l'ensemble d'apprentissage ou dans le processus de construction peut entraîner des changements importants dans le prédicteur résultant. Ces méthodes instables peuvent être améliorées au sens de leur performance par le principe du *Perturb & Combine* [Bre98].

L'idée du *Perturb & Combine*, décrite par la figure 3.2, consiste à générer plusieurs versions du prédicteur en perturbant l'ensemble des données d'apprentissage ou la méthode de construction du prédicteur. Ensuite, il faut combiner ces différentes versions pour en former un seul prédicteur. Nous utilisons la notation P&C (Perturb & Combine) pour désigner ce type d'approches. Ce principe de P&C a été initialement exploité dans le domaine de la classification et la régression avec les arbres de décision [Geu01, Ber09].

Dans le domaine de l'estimation de la densité, ce principe P&C a été appliqué par [Rid02].

Nous essayerons dans la suite de cette partie de définir un protocole pour la

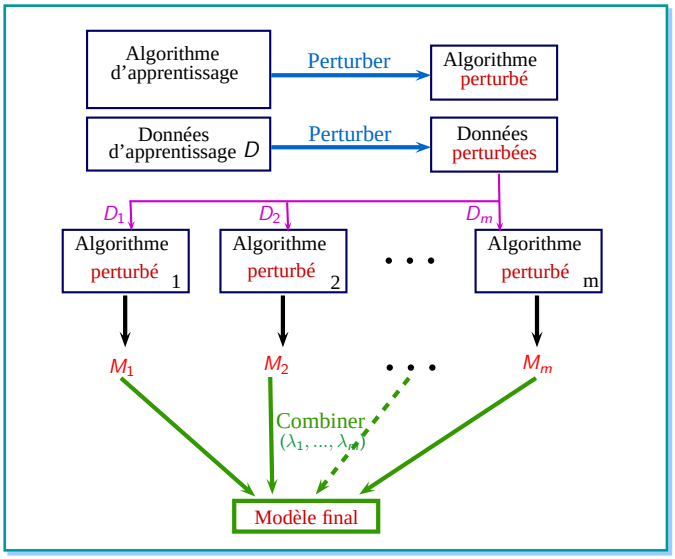


Figure 3.2. Principe du Perturbe & Combine

construction d'un mélange de modèles dans ce cadre de P&C.

Soit D un ensemble de données d'observations. La construction d'un mélange de m modèles peut être divisé en 4 étapes :

1. Génération des données D_i à partir des données observées
2. Génération des modèles M_i composant le mélange
3. Estimation des paramètres θ_i des modèles M_i constituant le mélange
4. Détermination des proportions λ_i du mélange

La perturbation sur ce protocole peut être considérée au niveau de plusieurs étapes.

- Un premier niveau : ***Perturbation des données*** où chaque élément (composant) du mélange est appris sur un ensemble de données d'apprentissage différent. Parmi les méthodes de manipulation des données d'apprentissage, nous retrouvons les principes de Bagging et de Boosting, dont nous parlons plus en détails dans les sections 3.4.2.1 et 3.4.2.2.
- Un deuxième niveau : ***Perturbation de la construction des modèles du mélange***. les méthodes de ce niveau s'intéressent plus particulièrement quant à elles à l'algorithme de génération des modèles élémentaires constituant le mélange. L'idée est généralement d'utiliser des paramétrages différents chaque fois qu'un modèle élémentaire est utilisé avec l'algorithme

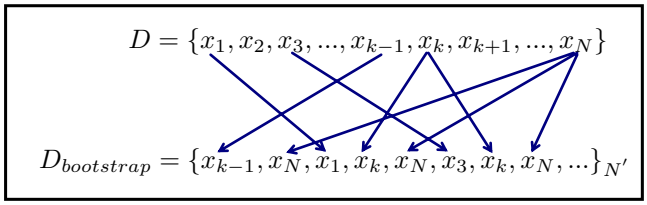


Figure 3.3. Principe du bootstrap

d'apprentissage. Par exemple, lorsqu'on génère un réseau bayésien, sa structure peut être apprise par l'un des algorithmes d'apprentissage de structure standard (certains nécessitant des initialisations pouvant être différentes ou aléatoires) ou échantillonnée aléatoirement.

Nous détaillons maintenant les différentes possibilités de perturber chacune de ces étapes.

3.4.2 Perturber les données d'apprentissage

3.4.2.1 Bagging

Le Bagging "Bootstrap aggregating" est une méthodes d'agrégation de prédicteurs introduite dans [Bre96a]. Elle consiste à générer plusieurs versions de prédicteurs et les agréger pour en former un seul. Chaque prédicteur est appris à partir d'un sous-ensemble de données (copie Bootstrap) généré à partir des données réelles par le principe du Bootstrapping [ET93]. Chaque prédicteur est alors appris sur un ensemble d'apprentissage différent.

Plusieurs versions de constitution des copies Bootstrap ont été proposées. A la base, pour construire une copie Bootstrap D_i à partir d'un ensemble de données d'apprentissage initial $D = \{x_1, x_2, \ldots, x_N\}$ de taille N, on tire uniformément et avec remise $N' \leq N$ exemples de D. D_i peut contenir des exemples dupliqués grâce au tirage avec remise. La figure 3.3 illustre un exemple de déroulement de ce principe.

Le Bagging a eu son succès dans le domaine de la classification [ZY08] et la régression. Cette technique permet de réduire la variance de l'estimateur final en combinant plusieurs modèles différents et d'éviter le sur-apprentissage. Elle permet aussi d'atténuer l'instabilité de modèles instables comme les arbres de décisions.

Plusieurs travaux ont utilisé le principe du Bagging. Citons [Ber09] qui a utilisé le Bagging avec la technique du Boosting pour l'induction des forêts aléatoires pour la classification.

Dans le contexte de l'estimation de densité, le Bagging de distributions normales a été proposé par [Rid02] pour contrôler la variance d'un modèle de mélange utilisant la technique du Boosting.

3.4.2.2 Boosting

Le Boosting a été introduit par [Sch90] comme une méthode incrémentale qui permet de combiner des modèles 'faibles' pour construire un 'bon' modèle final. Ce principe permet d'améliorer la performance de n'importe quel algorithme d'apprentissage.

AdaBoost (Adaptive Boosting), l'algorithme original du Boosting, a été proposé par [FS95] pour résoudre des problèmes de la classification. AdaBoost apprend séquentiellement des modèles faibles à partir d'une distribution des exemples d'apprentissage. D'une itération à l'autre, chaque exemple est pondéré en fonction de sa difficulté à être classé avec le classifieur courant. La perturbation au niveau de ce principe (comme c'est la cas dans le Bagging) est alors effectuée sur les données d'apprentissage de chaque modèle. Plusieurs variantes du Boosting ont été proposées, citons parmi ces variantes le stacking [Wol92] et l'arcing [Bre96b].

Le Boosting est largement utilisé dans le domaine de la classification avec différents types de classifieurs : les arbres de décision [Ber09], les réseaux bayésiens dynamiques [CRPP02], les réseaux Bayésiens naïfs [JPR08] et les réseaux de neurones [HPBM08].

Dans le contexte de l'estimation de densité, [Rid02] a proposé un algorithme de Boosting (et Bagging) de distributions normales. [RS02] a appliqué le principe du Boosting sur les réseaux bayésiens, comme modèles faibles, pour l'estimation de densité de probabilité. Dans le même cadre, [TSE02] ont appliqué ce principe sur des automates probabilistes fini.

3.4.3 Perturber la génération des modèles

3.4.3.1 Échantillonnage aléatoire

Pour pousser la diversité entre les modèles d'un mélange, il est possible d'utiliser une méthode d'échantillonnage aléatoire pour générer l'ensemble de ses composantes. Dans ce cas, les f_k sont générées par un échantillonnage dans tout l'espace des modèles possibles. L'échantillonnage dans l'espace des modèles peut être effectué selon une loi uniforme ou conformément à une autre loi comme l'utilisation des chaines de Markov. Dans ce cas, les paramètres des modèles f_k seront aléa-

toires ou appris à partir de données, qui peuvent eux aussi subir une opération de perturbation (voir section suivante).

Les paramètres du modèle de mélange (les proportions λ_i) peuvent aussi être aléatoires (uniformes par exemple) ou appris à partir des données par le principe du MV ou encore le MAP.

L'échantillonnage aléatoire de modèles a eu un succès énorme dans le cadre de la classification, notamment avec les forêts aléatoires initialement proposées par [Bre01]. Une forêt aléatoire est une combinaison d'arbres de décision construits à partir d'une réalisation d'un vecteur aléatoire de paramètres. Il a été montré que ces méthodes étaient particulièrement compétitives par rapport à l'un des principes d'apprentissage les plus efficaces, i.e. le Boosting.

3.4.3.2 Perturber un algorithme d'apprentissage

Il est possible de perturber un algorithme d'apprentissage en affaiblissant sa procédure de recherche, par exemple diminuer le degré d'optimisation en fixant un critère de convergence large, ou en fixant (générer aléatoirement) certains paramètres susceptibles de varier comme des initialisations. Ce type de perturbation ne peut pas être généralisé et reste spécifique à un algorithme donné. Par exemple, il est possible d'affaiblir un algorithme d'apprentissage à base de réseaux de neurones en générant aléatoirement les poids initiaux. Dans l'induction des arbres de décision, pour chaque noeud, un test est sélectionné parmi un ensemble de candidats. Il est alors possible de perturber cette procédure en choisissant aléatoirement un test parmi un ensemble des meilleurs candidats. Cette idée a été appliquée par [Die00] pour induire un ensemble d'arbres de décision.

3.5 Mélange de réseaux bayésiens

Lorsque l'espace des structures $\mathcal{M}$ et les paramètres Θ_M correspondant est l'espace des réseaux Bayésiens ou l'espace des réseaux de Markov sur X, des approximations doivent être faites pour rendre possible le calcul de l'équation (3.3). Par exemple pour les réseaux Bayésiens, [CH97] a montré que l'équation (3.2) peut être approchée par l'estimation de la vraisemblance avec les paramètres de probabilité maximum *a posteriori* $\tilde{\theta}_M = \arg\max_{\theta_M} P(\theta_M|M, D)$, avec une distribution de Dirichlet (paramétrée par ses coefficients α_i) pour la distribution a priori des paramètres $P(\theta_M|M)$.

Une autre approximation à considérer est celle permettant de simplifier la sommation sur toutes les structures M possibles. Comme la taille de l'ensemble des structures des modèles graphiques possibles est super-exponentielle en fonction du nombre de variables (voir section 2.6.3), la somme de l'équation (3.1) doit en pratique être calculée sur un sous-espace fortement contraint $\hat{\mathcal{M}}$ obtenu par exemple par des méthodes d'échantillonnage [MR94, MY95, FK00], rapportant ainsi l'approximation à

$$P_{\hat{\mathcal{M}}}(X|D) = \sum_{M \in \hat{\mathcal{M}}} P(M|D) P(X|\tilde{\theta}_M, M). \tag{3.6}$$

Notons ici que cette équation est encore une fois simplifiée lorsque des méthodes d'apprentissage de structure classiques sont utilisées, en ne retenant que le modèles $M = \tilde{M}$ qui maximise $P(M|D)$ sur $\mathcal{M}$:

$$P_{\tilde{M}}(X|D) = P(X|\tilde{\theta}_{\tilde{M}}, \tilde{M}). \tag{3.7}$$

3.6 Problèmes des espaces de grande dimension

Dans le cadre de grandes dimensions, les données sont relativement trop petites devant la taille du problème. Ce cadre favorise bien l'apparition du phénomène de sur-apprentissage déjà évoqué dans la section 1.5.

Le problème du sur-apprentissage survient le plus souvent lorsqu'on veut apprendre un modèle très complexe (grand nombre de paramètres ou variables) à partir d'un petit ensemble de données. Dans le contexte de notre travail, le sur-apprentissage est un vrai problème. Les mélanges de modèles faibles permettent ainsi de ne pas chercher à optimiser un modèle complexe mais essayer de faire l'estimation en utilisant un ensemble de plusieurs modèles moins complexe et facile à manipuler.

[HZ] a montré que, lorsque la taille des données est relativement petite, l'utilisation du *Bayesian Model Averaging* sur plusieurs ordres des noeuds permet d'améliorer la qualité de classification des réseaux bayésiens par rapport à l'utilisation d'un seul ordre sur ces noeuds.

Dans la pratique, on utilise généralement des modèles simples, soit parce qu'il n'y a pas assez de données pour estimer le «vrai» modèle (tous les paramètres d'un modèle), ou parce que les a priori disponibles sur le problème sont partiels.

La perturbation des données (par Bagging par exemple) permet aussi de pal-

lier à ce problème et introduire de la variabilité dans les données qui serviront à l'apprentissage de chaque élément du mélange, et par la suite introduire une diversité des modèles constituant le mélange. [Bre96b] a montré que le Bagging, avec des poids uniformes, évite le sur-apprentissage. De plus, le Bagging permet une amélioration de l'estimation grâce à la diversité qu'il engendre dans les modèles élémentaires.

Par contre, dans le cadre de la classification, [Dom00] a montré que le *Bayesian Model Averaging* n'évite pas le problème de sur-apprentissage présent lors d'une sélection de modèle et tend à l'aggraver. Cette tendance au sur-apprentissage dérive de la sensibilité de la vraisemblance aux fluctuations dans les données, et elle augmente avec le nombre de modèles considéré. Lorsque le Bagging est utilisé (avec des poids uniformes), chaque modèle est appris à partir d'un sous-ensemble des données initiales, ce qui élimine le phénomène de sur-apprentissage du modèle final.

L'apprentissage de mélanges de modèles en utilisant l'algorithme EM peut souffrir lui aussi des difficultés que peut rencontrer ce dernier dans l'espace de grande dimension. En effet, le risque de converger vers un optimum local est de plus en plus élevé dans ce cadre de grande dimension ce qui revient au phénomène de l'espace creux. Les solutions comme le lancement de l'algorithme d'apprentissage plusieurs fois à partir de différentes initialisations peut s'avérer coûteux et peu bénéfique devant la grande dimension de l'espace.

Un autre problème survient dans les espaces de grande dimension. Il est difficile de faire l'apprentissage de modèles complexes ayant un très grand nombre de variables (donc un grand nombre de paramètres à estimer) de point de vue complexité algorithmique et passage à l'échelle. En effet, plusieurs algorithmes d'apprentissage efficace en petite dimension s'avèrent très complexes et voir non réalisables dès que le nombre de variables dépasse un certain seuil. La technique des mélanges de modèles "simples" s'avère une bonne solution pour pallier à ce problème vu que ce sont des méthodes "scalables" aux grandes dimensions. Par exemple, l'apprentissage d'un réseau bayésien quelconque (DAG) est NP-difficile en fonction du nombre de variables tandis que l'apprentissage d'un arbre optimal peut être effectué efficacement et selon une complexité quadratique en fonction de la taille du problème.

3.7 Conclusion

Dans ce chapitre, nous avons présenté en premier temps le principe général des mélanges de modèles ainsi que le cadre général de cette approche. Nous avons ensuite présenté les différentes techniques d'apprentissage de mélanges de modèles de taille finie : des techniques basées sur l'algorithme EM, et d'autres basées sur le principe du *Perturb et Combine*.

Ensuite nous avons présenté un algorithme générique pour l'apprentissage de mélanges de modèles basé sur le principe du P&C en discutant les différentes possibilités d'instantiation de ses étapes. Nous avons ensuite présenté dans la section 3.5 les mélanges de réseaux Bayésiens.

La dernière section 3.6 a servi à discuter des problèmes des espaces de grande dimension qui se résument en deux types de problèmes théorique et pratique : le problème du sur-apprentissage et la "non scalabilité" des modèles complexes ayant un grand nombre de paramètres. Nous avons évoqué dans cette même section quelques solutions qui peuvent pallier à tous ces problèmes, notamment le principe du P&C et l'utilisation de modèles simples.

Pour toutes les raisons théoriques (bonne qualité d'estimation et pas de sur-apprentissage) et pratiques (scalabilité aux grandes dimensions) évoquées dans la section 3.6, nous nous orientons dans ce travail vers l'application du principe P&C à l'espace des modèles graphiques probabilistes pour apprendre des mélanges de modèles simples (arbres et polyarbres). Nous suivons le protocole décrit dans 3.4.1 pour apprendre un mélange de modèles simples et nous appliquons différentes stratégies pour perturber les données et/ou la procédure de génération des modèles composant le mélange.

Nous présenterons dans les deux chapitres suivants notre contribution dans ce domaine de l'estimation de la densité de probabilité par mélanges aléatoires de modèles graphiques probabilistes simples. Ce travail s'inscrit dans le cadre spécifique de la plupart des problèmes réels actuels, où le nombre d'observations disponible est très faible devant la très grande dimension du problème.

Chapitre 4

Mélanges aléatoires d'arbres pour l'estimation de densité

4.1 Introduction

Dans le cadre des problèmes de grande dimension (grand nombre de variables), nous nous intéressons à proposer des techniques d'estimation de la densité qui passent à l'échelle. Pour cela, nous proposons dans ce travail de tirer profit des avantages du principe P&C en l'appliquant dans le cadre de l'estimation de densité en utilisant les modèles graphiques probabilistes.

Nous considérons en premier temps des mélanges de modèles simples (arbres et polyarbres) générés aléatoirement selon une loi uniforme dans l'espace des modèles possibles. L'apprentissage du mélange ne se fait pas par l'algorithme itératif EM classiquement utilisé, mais selon une méthode directe, décrite dans la section 4.2, où les paramètres sont estimés séparément pour chaque modèle. Quand aux paramètres relatifs aux proportions du mélange, ils sont estimés indépendamment des autres paramètres.

Nous commençons dans la première section de ce chapitre par présenter les propriétés des polyarbres ainsi que les mélanges de ces modèles simples. Nous présentons ensuite notre algorithme général pour l'apprentissage d'un mélange de modèles simples. La section 4.3 est consacrée à décrire différentes variantes de cet algorithme général en exploitant le principe du P&C ainsi qu'une étude de la complexité de cet algorithme général.

Ces différentes variantes ont fait le sujet de plusieurs publications scientifiques. Une première publication [ALDW08b] à été présentée dans *The Annual Belgian-*

Dutch Machine Learning Conference. La méthode décrite dans la section 4.3.1 a fait le sujet d'une seconde publication ([ALW08]) dans les 4èmes Journées Francophones sur les Réseaux Bayésiens, ensuite elle a été mieux détaillée dans une autre publication ([ALDW08a]) lors du 4ème workshop européen *The European Workshop on Probabilistic Graphical Models*.

La méthode décrite dans la section 4.3.2 a été publiée ([ALDW09b]) dans la 10ème conférence européenne *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, et présentée ([ALDW09a]) lors de la *Conférence francophone sur l'APprentissage artificiel*.

Dans la section 4.4, nous présentons un protocole expérimental que nous avons suivi pour tester et valider notre approche. Nous concluons ce chapitre par une présentation et une discussion de nos résultats pour cette proposition.

4.2 Algorithme général

Nous avons vu dans le chapitre 2 que les structures arborescentes (arbres, polyarbres) ont l'avantage de pouvoir effectuer efficacement les algorithmes d'inférence et d'apprentissage selon une complexité raisonnable. Pour cela, nous proposons d'utiliser ces modèles simples pour répondre aux besoins du cadre dans lequel nous travaillons (espaces de grande dimension).

4.2.1 Les polyarbres et leurs propriétés

Un polyarbre est défini par une structure de graphe P sans circuits et dirigée dont le squelette est sans cycles et connecté, l'ensemble des noeuds en bijection avec $X = \{X_1, \ldots, X_n\}$ et l'ensemble des densités conditionnelles $\mathbb{P}_P(X_i|pa_P(X_i))$, où $pa_P(X_i)$ représente l'ensemble des variables en bijection avec les parents de X_i dans P.

La structure P peut être exploitée pour l'inférence probabiliste par $\mathbb{P}_P(X)$ en utilisant l'algorithme de *Messages locaux* avec une complexité algorithmique linéaire en fonction du nombre de variables n (voir section 2.3.2).

4.2.2 Les mélanges de polyarbres

Une distribution de mélange $P_{\hat{\mathcal{P}}}(X_1, \ldots, X_n)$ sur un ensemble $\hat{\mathcal{P}} = \{P_1, \ldots, P_m\}$ de m polyarbres est définie comme une combinaison convexe de

Algorithme 8 Algorithme : Apprentissage de mélanges de polyarbres (structures arborescentes)

1: **Pour** $i = 1$ to m **Faire**
2: $D_i = DrawData(D)$,
3: $T_i = DrawPolytree$,
4: $\tilde{\theta}_{T_i} = LearnPars(T_i, D)$
5: **Fin Pour**
6: $(\lambda_i)_{i=1}^m = CompWeights((T_i, \tilde{\theta}_{T_i})_{i=1}^m, D)$
7: **return** $\left(\lambda_i, T_i, \tilde{\theta}_{T_i}\right)_{i=1}^m$.

densités élémentaires de polyarbres, ie.

$$P_{\hat{\mathcal{P}}}(X) = \sum_{i=1}^{m} \lambda_i P_{P_i}(X), \tag{4.1}$$

où $\lambda_i \in [0, 1]$ et $\sum_{i=1}^m \lambda_i = 1$.

Même si les modèles de polyarbres imposent des restrictions fortes sur le type des densités qu'ils peuvent fidèlement représenter, les mélanges de polyarbres, comme les mélanges d'arbres, sont des approximateurs universels.

4.2.3 Apprentissage de mélanges aléatoires de polyarbres

Notre procédure générique pour apprendre une distribution aléatoire de mélange de polyarbres à partir d'ensemble de données D est décrite dans l'algorithme 8. Le principe de cet algorithme est basé sur le principe du P&C décrit dans la section 3.4.1 d'une part, et sur les méthodes d'échantillonnage décrites dans la section 3.4.3 d'autre part. Ces choix reviennent à plusieurs raisons. D'abord, le principe du Perturb nous permet d'échapper au phénomène du sur-apprentissage assez fréquent lorsque l'on veut optimiser un estimateur complexe à partir d'un ensemble faible de données. De plus, échantillonner dans l'espace de recherche nous permet de réduire la complexité algorithmique de nos mélanges (nous n'aurons pas à apprendre la structure d'un modèle) ce qui facilitera le passage à l'échelle de nos méthodes.

Cet algorithme général, décrit dans l'algorithme 8, reçoit en entrée les variables X, les données d'apprentissage D, la taille maximale du mélange m et quatre procédures *DrawData*, *DrawPolytree*, *LearnPars*, *CompWeights*.

A l'itération i, la procédure *DrawData* permet de générer un ensemble de

données D_i à partir de l'ensemble initial D. La procédure *DrawPolytree* génère une structure T_i de polyarbre et la procédure *LearnPars* est relative à l'apprentissage des paramètres de la structure T_i. La quatrième procédure *CompWeights* retourne les coefficients de pondération λ_i de tous les modèles.

Il est possible d'appliquer le principe P&C au sein de cet algorithme à différents niveaux. En effet, la procédure *DrawData* peut être perturbée par exemple en appliquant l'une des méthodes de perturbation des données d'apprentissage décrites dans la section 3.4.2 (Bagging, Boosting, etc). De plus, la procédure *DrawPolytree* permet de perturber la génération des modèles du mélange en la randomisant (voir section 3.4.3). La procédure *CompWeights* permet ensuite de calculer, selon différentes politiques, les coefficients du mélange qui serviront pour combiner les modèles simples et en former un seul.

En ce qui concerne l'apprentissage des paramètres d'une structure T_i par la procédure *LearnPars* à partir de données, nous utilisons le principe de la méthode bayésienne MAP décrite dans la section 1.3.3 avec des a priori uniformes de Dirichlet. Ce choix revient au fait que nous ne disposons que de très peu d'échantillons en comparaison avec le nombre de variables du problème.

Nous avons exploité, dans ce travail, ces différents degrés de liberté. Plusieurs variantes sont étudiées dans la section 4.3. En effet, nous présentons dans la section 4.3.1 deux types de mélanges de modèles simples où le principe du P&C a été appliqué sur la procédure de génération de modèles simples, le premiers formé par un ensemble de polyarbres, tandis que le deuxième est formé par un ensemble d'arbres dirigés.

La section 4.3.2 est consacrée à présenter un autre type d'instanciation de notre algorithme général en appliquant le principe du P&C pour perturber les données d'apprentissage. Nous étudierons dans cette section l'apport de la perturbation de l'ensemble de données d'apprentissage tout en optimisant la procédure de génération des composants du mélange.

La dernière variante de notre algorithme général est décrite dans la section 4.3.3. Cette perturbation porte sur la procédure qui permet de générer les coefficients de pondération des composants du mélange.

Ces différentes instanciations sont ensuite évaluées par le moyen d'un protocole expérimental en comparaison avec différentes méthodes de l'état de l'art, notamment les méthodes décrites dans la section 3.3.3.

4.3 Variantes de l'algorithme général

L'algorithme générique 8 est constitué principalement de quatre procédures. Chacune de ces procédures peut subir une opération de perturbation.

Nous détaillons dans la suite un ensemble de politiques de perturbation de chacune de ces procédures et nous étudions l'impact de ces perturbations sur la qualité du modèle final.

4.3.1 Génération aléatoire de structures arborescentes (arbres et polyarbres)

4.3.1.1 Généralités

Nous proposons dans un premier temps deux variantes pour la procédure *DrawPolytree*. La première (*DrawRndPolytree*) consiste à générer aléatoirement un ensemble de polyarbres pour construire le mélange de modèles simples. La seconde (*DrawRndTree*) consiste à se restreindre à l'espace des arbres (dirigés).

Dans ce cas, la procédure *DrawData* de l'algorithme général 8 n'a pas de sens puisqu'on ne perturbe pas les données. Cette procédure est alors une fonction identité qui renvoie l'ensemble de données initial D à D_i.

La génération des modèles de polyarbres ou des arbres se fait selon une loi uniforme. La procédure de génération de ces modèles simples est celle décrite dans [Qui89]. Elle se base sur les propriétés de codage de Prûfer pour les structures d'arbres. Prûfer a établit une bijection entre l'ensemble des arbres complètement étiquetés à n noeuds et l'ensemble de listes de $(n-2)$ nombres entiers dans $\{1, 2, \ldots, n\}$ (répétition autorisée)

Pour construire un arbre non dirigé à n noeuds, les étapes sont décrites dans l'algorithme 9.

L'algorithme 9 nous permet de générer une structure d'arbre non orientée selon une loi uniforme. En effet, la probabilité de générer aléatoirement un entier dans $\{1, 2, \ldots, n\}$ est égale à $1/n$. La probabilité d'une liste de taille $n-2$ dans cet ensemble est alors $p = 1/n^{n-2}$. Le théorème de Cayley montre que le nombre d'arbres à n noeuds est n^{n-2}. Comme il existe une bijection entre les listes à $(n-2)$ nombres entiers et les arbres à n noeuds, notre génération se fait donc selon une loi uniforme.

L'orientation des arcs de ces arbres étiquetés peut se faire en utilisant un algorithme classique de parcours en profondeur. Nous utilisons pour cela un algorithme

Algorithme 9 Algorithme : Génération de structure arborescente non orientée

Entrées: n {*nombre de variables (noeuds)*}

Pour $i = 1, \ldots, n-2$ **Faire**
Générer un entier a_i uniformément dans $\{1, 2, \ldots, n\}$
$a \leftarrow a \cup \{a_i\}$
Fin Pour
$b \leftarrow (1, 2, \ldots, n)$
Tantque $a \neq \emptyset$ **Faire**
Chercher t_1 le plus petit entier dans b non dans a,
Joindre a_1 à t_1 pour former une arête dans l'arbre T
$a \leftarrow a \setminus \{a_1\}$
$b \leftarrow b \setminus \{t_1\}$
Fin Tantque
Joindre les deux entiers restant dans b pour former la dernière arête de l'arbre T

return T

qui nous fournit un ordre de visite sur les arcs du graphe en partant d'un noeud racine. Le noeud racine est tiré au hasard parmi les n noeuds du graphe.

4.3.1.2 Génération aléatoire et uniforme d'arbres (DrawRndTree)

Pour générer un arbre orienté, nous utilisons l'algorithme 9 pour générer une structure non orientée. Ensuite, l'orientation des arcs se fait en utilisant un algorithme classique de parcours en profondeur qui nous fournit un ordre de visite sur les arcs du graphe en partant d'un noeud racine. Le noeud racine est tiré au hasard parmi les n noeuds du graphe. Cette orientation se fait dans un seul sens (du noeud dont l'indice est le plus petit vers celui dont l'indice est le plus grand), ceci pour garantir une structure d'arbre sans V-structures.

Cette démarche nous garantit un échantillonnage uniforme dans l'espace des arbres dirigés. En effet, la génération de la structure est uniforme dans l'espace des arbres non orientés. De plus, la procédure d'orientation de cette structure se fait d'une façon déterministe une fois le noeud racine est choisi. Il existe alors n possibilités pour transformer un arbre non orienté de n noeuds en un arbre orienté. Comme nous choisissons le noeud racine aléatoirement et selon une loi uniforme de probabilité égale à $1/n$, alors cette procédure d'orientation d'une structure est uniforme aussi.

En conclusion, la démarche de générer un arbre orienté se fait selon une loi

uniforme dans l'espace des arbres (dirigés).

Il est à remarquer aussi que cette procédure est aussi uniforme dans l'espace des classes d'équivalence de Markov relatives aux structures d'arbres. En effet, une classe d'équivalence de Markov relative à des structure d'arbre est représentée par l'arbre non orienté généré par la procédure décrite dans l'algorithme 9 selon une loi uniforme. Donc la génération de la structure non orientée correspond à la désignation de la classe d'équivalence de l'arbre orienté final.

4.3.1.3 Génération aléatoire et uniforme de polyarbres (DrawRndPolytree)

Pour générer un polyarbre, nous utilisons l'algorithme 9 pour générer une structure arborescente non orientée. Ensuite, les arêtes de l'arbre non orienté sont parcourues en utilisant un algorithme de parcours en profondeur en partant d'un noeud racine. Cette fois ci, l'orientation d'un arc n'est pas donnée par le sens de parcours mais par une loi aléatoire uniforme ($p = 1/2$).

Comme l'orientation des arcs se fait d'une façon indépendante pour chaque arc et selon une loi uniforme, la procédure d'orientation d'un arbre non dirigé en un polyarbre est également faite selon une loi uniforme pour un arbre non dirigé fixé. En effet, le nombre total de polyarbres qui peuvent être générés à partir d'un arbre non orienté est alors $n\ 2^{(n-1)}$, et la probabilité d'avoir un polyarbre est de $1/n2^{(n-1)}$.

En conclusion, la génération d'un polyarbre orienté se fait selon une loi uniforme dans l'espace des polyarbres.

Contrairement aux cas des arbres dirigés, la génération des polyarbres n'est pas uniforme dans l'espace des représentants des classes d'équivalence de Markov. L'exemple de la figure 4.1 montre que deux classes d'équivalences relatives à un même nombre de variables sont de tailles différentes. La première classe peut contenir jusqu'à quatre polyarbres équivalents au sens de Markov, tandis que la deuxième ne peut contenir qu'un seul polyarbres. Cet exemple montre bien que la deuxième classe d'équivalence est moins probable que la deuxième à cause de sa taille réduite.

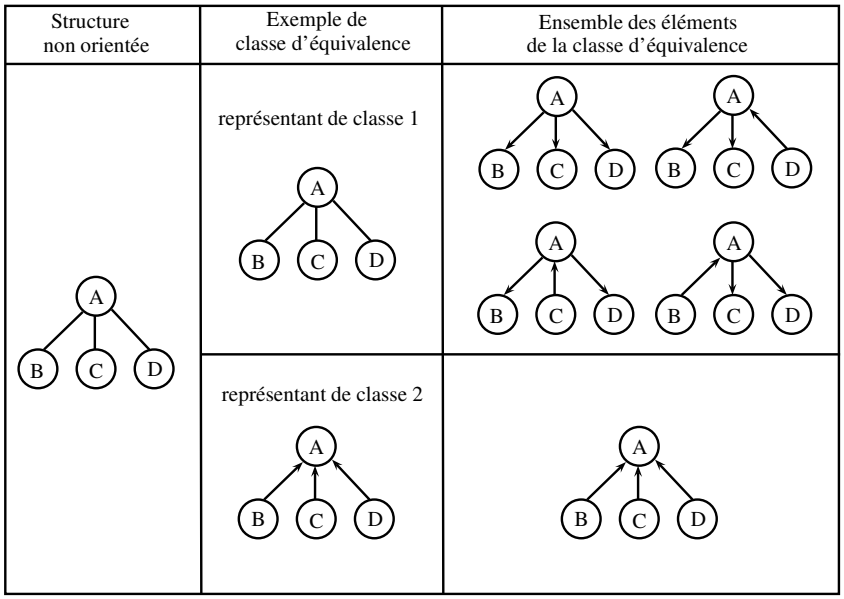


Figure 4.1. Exemple de deux classes d'équivalences de Markov possibles pour une même structure non orientée, ainsi que les différents éléments de chaque classe d'équivalence de Markov

4.3.2 Perturbation des données d'apprentissage par Bootstrapping

Nous proposons dans cette section une nouvelle variante pour la génération des modèles simples constituant le mélange. Dans la section 4.3.1, le principe du Perturb est appliqué à la génération aléatoire d'une structure (d'arbre ou de polyarbre). Nous proposons maintenant d'appliquer le principe du P&C sur les données d'apprentissage. La technique de perturbation des données utilisée est le *Bootstrapping* décrite dans la section 3.4.2.1. Le but de cette perturbation des données est de tirer profit des avantages du Bagging dans le cas où le nombre d'échantillons disponibles est limité en comparaison avec la taille du problème.

La génération des modèles du mélange ne se fait plus aléatoirement mais à l'aide de l'algorithme MWST, décrit par l'algorithme 3 de la section 2.8.2, sur les données Bootstrappés.

Le principe de cette variante est décrit par l'algorithme 10.

Le principe de l'algorithme 10 est le même que l'algorithme générique 8. La procédure *DrawData* correspond dans ce cas à la génération d'une copie bootstrap D_i à partir de D (*BootstrapReplica*), et la procédure de génération du modèle T_i correspond à la recherche de l'arbre de poids maximal (algorithme MWST) sur les données D_i. La procédure *DrawPolytree* dans l'algorithme 8 est alors remplacée

Algorithme 10 Algorithme : Apprentissage de mélanges d'arbres de Markov

Pour $i = 1$ to m **Faire**
$D_i \leftarrow BootstrapReplica(D)$,
$T_i = LearnMWST(D_i)$,
$\tilde{\theta}_{T_i} = LearnPars(T_i, D_i)$
Fin Pour
$(\lambda)_{i=1}^{m} = CompWeights((T_i, \tilde{\theta}_{T_i})_{i=1}^{m}, D)$
return $\left(\lambda_i, T_i, \tilde{\theta}_{T_i}\right)_{i=1}^{m}$.

par la procédure $LearnMWST$ dans l'algorithme 10.

4.3.3 Calcul des coefficients du mélange

Nous avons vu dans la section 3.3.1 que les proportions du mélange peuvent être appris itérativement par l'algorithme EM à partir des données conjointement avec les paramètres des composantes du mélange. Nous proposons dans notre algorithme général d'estimer directement ces paramètres. Il est possible d'appliquer à l'extrême le principe du P&C sur cette étape de notre algorithme générique et imaginer une génération aléatoire des poids du mélange. Nous proposons dans cette partie d'utiliser, étudier et comparer deux schémas de pondérations (moins aléatoires) : un premier *schéma uniforme* et un second *schéma Bayésien* calculé en fonction de la probabilité a posteriori des modèles composant le mélange sachant les données D. Ces derniers poids sont notés $BDeu$, puisqu'ils sont calculés en fonction du score $BDeu$ de chaque modèle T_i (voir section 2.6.3).

Dans le cas où les coefficients sont uniformes, la procédure *CompWeights* correspond à attribuer un poids uniforme $\lambda_i = 1/m$ pour chaque élément T_i du mélange, avec m est la taille du mélange.

Les poids $BDeu$ sont calculés de la façon suivante :

$$\lambda_i = ScoreBDeu(T_i, D) / \sum_{k=1}^{m} ScoreBDeu(T_k, D)$$

où le scoreBDeu est donné par l'équation 2.5 dans le cas discret.

La normalisation par la somme des scores est nécessaire pour avoir une somme des poids égale à 1.

Pour des raisons pratiques, nous remplaçons le scoreBDeu par la quantité suivante : $\exp(ScoreBDeu(T_i, D) - maxScores)$, où $maxScores$ est le score le plus

élevé pour les modèles du mélange. Cette nouvelle formule du scoreBDeu nous permet de privilégier le meilleur modèle dans le mélange en lui attribuant un poids très important par rapports aux autres modèles.

Le premier schéma de pondération donne le même poids pour tous les modèles. Dans le deuxième, on récompense les meilleurs modèles au sens de probabilité a posteriori. Le score $BDeu$ permet donc de mesurer la qualité d'un modèle sachant les données.

4.3.4 Complexité en temps et en espace

Puisque nous nous intéressons en particulier aux domaines de grande dimension (grand nombre de variables), la complexité algorithmique et espace mémoire doivent être considérés et étudiés pour garantir le passage à l'échelle de nos méthodes et préciser leurs limites. On ne s'intéresse alors qu'a l'étude de la complexité en fonction de n.

4.3.4.1 Complexité en temps

Nous commençons par étudier la complexité de chacune des procédures de l'algorithme général, ensuite nous en déduisons la complexité totale de chacune des variantes.

La première procédure de notre algorithme général *DrawData* n'est pas utilisée lorsque nous n'effectuons pas de perturbation sur les données d'apprentissage (section 4.3.1). Dans le cas des méthodes à base de Bagging (section 4.3.2), la génération d'une copie bootstrap à partir des données initiales nécessite exactement la génération aléatoire de N' nombres aléatoires entre 1 et N (version standard du bootstrap), donc une complexité linéaire en fonction du nombre d'échantillons dans la base de données initiale. Elle ne dépend pas du nombre de variables.

La génération aléatoire d'arbres (polyarbres) (variante *DrawRndPolytree* et *DrawRndTree*) est de complexité algorithmique linéaire en fonction du nombre de variables considérées donc $O(n)$. La recherche de l'arbre de poids maximal en utilisant l'algorithme MWST est de complexité quadratique, $O(n^2 \log(n))$.

L'apprentissage des paramètres d'une structure arborescente est aussi linéaire en fonction du nombres de variables et du nombre de données, donc une complexité $O(n)$.

En ce qui concerne le calcul du poids de chaque élément dans le mélange, un poids uniforme nécessite une seule opération de division $(1/m)$, tandis que le calcul

Table 4.1. Complexité des différentes procédures de l'algorithme général

Procédure	**complexité**
DrawData	$O(1)$
DrawPolytree	$O(n)$
LearnMWST	$O(n^2 \log(n))$
LearnPars	$O(n)$
CompWeightsUnif	$O(1)$
CompWeightBDeu	$O(n)$

du score BDeu de la structure est linéaire en fonction de n comme il s'agit d'un score décomposable.

Le tableau 4.1 récapitule la complexité des différentes procédures de notre algorithme général ainsi que les différentes variantes.

La construction d'un mélange d'arbres (ou de polyarbres) aléatoires va utiliser m fois les procédures de l'algorithme général, et sera donc de complexité $O(mn)$.

De même, la construction d'un mélange d'arbres baggés utilise m fois les procédures appropriées de l'algorithme général, et sera donc de complexité $mn^2 \log(n)$.

Le seul point bloquant pour ces variantes est alors la procédure *LearnMWST* qui est quadratique en fonction de n.

4.3.4.2 Complexité en espace

Tout comme la complexité algorithmique, les demandes de nos approches en l'espace mémoire est un critère important pour étudier le passage à l'échelle de ces approches ainsi que leurs limites.

Nous commençons par calculer l'espace mémoire nécessaire pour chaque procédure de notre algorithme général, nous en déduisons par la suite les demandes de chaque variante.

- La première procédure *DrawData* nécessite, dans le cas où nous générons une copie Bootstrap des données initiaux, le stockage de n nombres entiers, donc une complexité en espace $O(n)$.
- Les procédures *DrawRndTree* et *DrawRndTree* nécessitent le stockage de $n-2$ entiers pour stocker la structure du modèle, donc une complexité en espace $O(n)$.
- La procédure *LearnMWST* nécessite un stockage de $n(n-1)/2$ réels (pour remplir la matrice symétrique d'information mutuelle), donc une complexité en espace $O(n^2)$.

- La procédure *LearnPars* nécessite un nombre de paramètres réels linéaire en fonction de n dans le cas d'un arbre, et un nombre de paramètres au pire quadratique dans le cas d'un polyarbre.
- La procédure *CompWeights* ne nécessite qu'une seule valeur réelle.

L'espace mémoire total nécessaire pour construire un mélange d'arbres aléatoires de taille m et alors $O(mn)$. Alors que la construction d'un mélange de m polyarbres aléatoires ou m arbres baggés est de complexité $O(n^2)$.

Deux points bloquant pour nos méthodes de point de vue complexité en espace. Le premier est relatif à l'apprentissage de paramètres d'un polyarbre qui peut atteindre une complexité $O(n^2)$, mais cela peut être contrôlé en fixant un nombre maximum de parents possibles par noeud. Le deuxième point bloquant est relatif au calcul de la matrice d'information mutuelle.

4.4 Validation expérimentale

4.4.1 Protocole expérimental

Pour valider notre approche et mesurer la qualité de l'estimation de nos mélanges de modèles simples dans le cadre de grande dimension (plusieurs variables) et très peu de données, nous avons testé les différentes variantes de notre algorithme général sur un ensemble de modèles cibles de complexité croissante (chaînes, arbres, polyarbres, DAG). Ces modèles cibles serviront de modèles théoriques pour évaluer la qualité de l'estimation de nos méthodes. Ils serviront aussi pour la simulation de données d'apprentissage.

Principe de génération de structures

Comme nous nous intéressons au cadre de grande dimension, nous avons utilisé des distributions cibles de taille élevée ($1000, 5000$ variables).

En ce qui concerne la génération des modèles cibles, nous avons utilisé la méthode décrite dans la section 4.3.1 pour la génération de chaînes, arbres et polyarbres, et la méthode proposée dans [ICR04] pour la génération de structures de DAG.

Dans ce travail, nous considérons seulement des variables discrètes et binaires. En ce qui concerne la génération des paramètres d'une distribution cible, pour chaque densité conditionnelle de la structure, nous générons aléatoirement deux nombres dans l'intervalle $[0, 1]$ en les normalisant pour obtenir deux probabilités qui somment à 1.

	1000 var.	5000 var.
100 éch.	X	
250 éch.	X	X
1000 éch.	X	

Table 4.2. Taille des ensembles de données utilisés pour chaque dimension des distributions cibles considérées lors des simulations

Génération des données d'apprentissage

Nous avons considéré différentes tailles de bases de données d'apprentissage, avec un nombre d'individus relativement petit devant la dimension du modèle cible. Pour les distributions cibles de taille $n = 1000$, nous avons testé des bases de données de taille $N = 100, 250$ et 1000. Le tableau 4.2 présente les différentes tailles des données utilisées pour chaque dimension de l'espace considéré.

Pour chaque distribution cible, et pour chaque taille des données, nous générons 10 bases de données différentes en échantillonnant selon la loi de la distribution cible et ses paramètres. L'échantillonnage des données est fait en utilisant la méthode de Monte Carlo décrite dans la section 2.5.5.

Taille du mélange

Contrairement aux méthodes de mélanges basées sur l'algorithme EM où la taille m du mélange doit être connue ou apprise par l'algorithme EM, nous utilisons un mélange de modèles simples de taille croissante, respectivement $m = 1$, $m = 10$, et allant jusqu'à $m = 150$. Cela nous permet de faire apparaître l'effet de cette variables sur la qualité de l'estimation du modèle final.

Différentes méthodes testées

L'application du principe P&C sur l'algorithme général a donné naissance à différentes variantes que nous résumons dans le tableau 4.3. En plus de ces différentes variantes, nous avons testé, dans les mêmes conditions et en suivant la même démarche, cinq autres méthodes de l'état de l'art. Les méthodes de références utilisées sont les suivantes :

- La méthode nommée *GO*, obtenue en utilisant la structure cible et en réestimant ses paramètres à partir des données d'apprentissage. Cette méthode représente la meilleure solution de recherche d'un seul modèle pour estimer le modèle cible puisqu'elle utilise la structure de la distribution cible ayant généré les données.
- La méthode de base *BL*, naïve qui estime directement la loi jointe sans

Table 4.3. Les différentes variantes de l'algorithme général considérées ainsi que les méthodes de base

Nom	**Génération des structures**	**Données**	**Poids**	**Complexité en temps**
MTU	DrawRndTree	D	uniformes	$O(mn)$
MTBDe	DrawRndTree	D	BDeu	$O(mn)$
MPTU	DrawRndPolytree	D	uniformes	$O(mn)$
MPTBDe	DrawRndPolytree	D	BDeu	$O(mn)$
CLBU	DrawMWST	B	uniformes	$O(mn^2 \log(n))$
CLBBDe	DrawMWST	B	BDeu	$O(mn^2 \log(n))$
GO	Structure cible	D	1	$O(n)$
NBE*	Naîf Bayes	D	EM	$O(mn)$
MixTree	DrawMWST	D	EM	$O(mn^2 \log(n))$
CL	DrawMWST	D	1	$O(n^2 \log(n))$

hypothèse d'indépendance entre les variables. Cela revient à construire une structure de graphe complet dont les paramètres sont estimés à partir de l'ensemble de données D.

- La méthode nommée *NBE**, obtenue en utilisant une série de modèles de Bayes Naïf avec une taille croissante de la variable classe construits par le moyen de l'algorithme EM. Ce modèle de Bayes Naïf (décrit par l'algorithme 6 dans la section 3.3.3.1) est proposé par [LD05] en intégrant une phase d'élagage que nous n'utilisons pas pour se mettre dans le même contexte que nos méthodes où nous n'utilisons pas de tels raffinements (élimination des mauvais éléments du mélange).
- La méthode *MixTree* proposée par [MP99] et décrite dans la section 3.3.3.2. Dans cette méthode, le mélange est constitué par un ensemble d'arbres optimaux, et est construit en combinant l'algorithme MWST avec l'algorithme EM pour l'apprentissage des paramètres du mélange.
- La méthode *CL*, obtenue en appliquant l'algorithme MWST sur l'ensemble de données initial.

Mesure de la qualité de l'estimation

Pour mesurer la qualité d'estimation de chaque densité de probabilité apprise à partir des données, nous évaluons la divergence asymétrique de Kullback-Leibler (KL) [KL51] entre cette densité et celle $P_G(X)$ qui a été utilisée pour générer les données.

Le calcul exact de cette quantité est donné par :

$$KL(P_G, P_M) = \sum_{X \in \mathcal{X}} P_G(X) \ln \frac{P_G(X)}{P_M(X)}, \tag{4.2}$$

où $P_M(X)$ représente la densité estimée que nous voulons évaluer, et $\mathcal{X}$ représente l'ensemble de toutes les configurations possibles des variables aléatoires dans X.

Comme nous considérons un grand nombre de variables, la taille de $\mathcal{X}$ est exponentiellement grande en fonction du nombre de variables (pour des variables binaires, et $n = 1000$, $|\mathcal{X}| = 2^{1000}$). Nous avons alors recours à un calcul approché de la divergence de KL en utilisant la méthode du Monte Carlo et le principe de l'échantillonnage préférentiel [Sri01]. L'estimation de KL approchée est donnée par :

$$\hat{KL}(P_G, P_M) = \sum_{X \sim P_G} P_G(X) \ln \frac{P_G(X)}{P_M(X)}, \tag{4.3}$$

Pour le cas de problèmes de dimension 1000 (resp. 5000), nous avons utilisé 5000 (resp. 100000) échantillons pour chaque estimation Monte Carlo de la divergence de KL.

Cette mesure s'interprète comme suit : plus $\hat{KL}(P_G, P_M)$ est petite, plus l'estimation de P_G par P_M est bonne. Autrement dit, si $\hat{KL}(P_G, P_{M1}) < \hat{KL}(P_G, P_{M2})$, alors l'estimation de P_G par P_{M1} est meilleure que son estimation par P_{M2}. Si la divergence de KL est nulle, P_M coïncide avec P_G.

En plus de la divergence de KL, nous évaluons aussi nos méthodes de point de vue du temps de calculs.

Solutions logicielles

Pour la réalisation de ce protocole expérimental, nous avons utilisé le langage de programmation ***C/C++*** ainsi qu'un ensemble de bibliothèques qui nous ont facilité un certain nombre de tâches. La bibliothèque ***Boost***[1] nous a fournis un environnement pour manipuler les graphes. La bibliothèque ***ProBT***[2] spécialisée en programmation bayésienne nous a permis de manipuler les lois de probabilités, modéliser les lois jointes ainsi qu'un certain nombre d'algorithmes d'inférence et d'apprentissage de paramètres. Le package *StructureLearning*, développé au sein de notre équipe et ensuite introduit dans la librairie *ProBT*, nous a permis de manipuler des réseaux bayésiens et d'apprendre la structure grâce à un nombre

1. http ://www.boost.org/
2. http ://emotion.inrialpes.fr/BP/spip.php ?rubrique6

d'algorithmes à base de scores.

4.4.2 Résultats

Nos résultats sont présentés par le moyen de figures représentant la qualité de l'estimation (divergence de KL) en fonction de la taille du mélange. Ces résultats sont des moyennes pour 10 distributions cibles de type DAG, et pour 10 bases de données de différentes tailles pour chaque distribution cible. Les courbes en trait continu sont relatives à un schéma de pondération bayésien (en fonction du score BDeu de la structure sachant les données), tandis que les courbes en trait discontinu sont relatives à des poids unifomes du mélange.

4.4.2.1 Qualité d'estimation

Mélanges d'arbres et mélanges de polyarbres

La figure 4.2 représente les résultats obtenus pour les deux méthodes de mélanges aléatoires d'arbres (courbes en rouge) et de polyarbres (courbes en bleue).

D'après la figure 4.2, nous observons que *les mélanges aléatoires d'arbres (MTU et MTBDe) fournissent des résultats équivalents à ceux obtenus par les mélanges aléatoires de polyarbres (MPTU et MPTBDe)*, malgré le fait que les polyarbres soient des modèles plus riches en pouvoir de modélisation grâce aux indépendances conditionnelles (V-structures) qu'ils peuvent représenter. Ce premier résultat a été observé aussi dans le cadre d'ensembles de données de taille plus grande (250 et 1000), cela rejoint le fait que les mélanges d'arbres et de polyarbres sont des approximateurs universels.

Nous avons montré dans la section 4.3.1.3 que la génération des polyarbres ne se fait pas selon une loi uniforme dans les classes d'équivalences de Markov relatives à des polyarbres. Nous avons vu aussi que les classes d'équivalences de Markov dont le graphe représentant contient le moins de V-structures sont de taille plus grande que ceux dont le graphe représentant contient plusieurs V-structures, ce qui rend élevée la probabilité qu'un polyarbre généré aléatoirement appartient à une classe d'équivalence dont le nombre de V-structures est faible. Autrement dit, un polyarbre généré aléatoirement a plus de chance d'appartenir à une classe d'équivalence de Markov avec peu de V-structures ce qui est proche des arbres dirigés.

Le résultat observé précédemment peut alors être expliqué par le faite que les mélanges de polyarbres contiennent un grand nombre d'arbres et de polyarbres

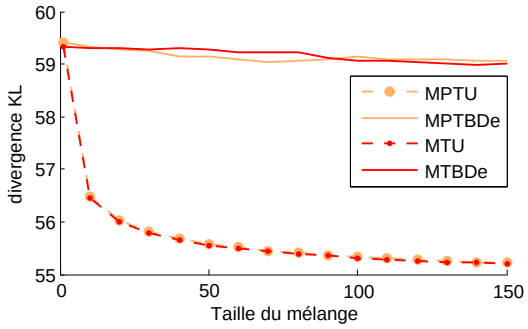


Figure 4.2. Mélanges aléatoires d'arbres (courbes rouges) et mélanges aléatoires de polyarbres (courbes oranges) pour 10 DAG de 1000 variables et des données de taille 100

avec très peu de V-structures.

Mélanges d'arbres baggées

La figure 4.3 se place dans le cas de mélanges d'arbres aléatoires MT en rajoutant les résultats obtenus pour des mélanges d'arbres baggés CLB (arbres optimaux sur des copies bootstrap) ainsi que la méthode de base CL où un seul arbre optimal est construit en utilisant l'algorithme MWST sur les données initiales.

Nous rappelons que les mélanges d'arbres baggés sont constitués par des arbres optimisés en utilisant l'algorithme MWST sur des copies bootstrap D_i des données initiales D (voir section 4.3.2).

D'après la figure 4.3, nous remarquons que *ces méthodes à base du Bagging sont nettement meilleures que les mélanges aléatoires d'arbres*, ceci avec toutes les tailles des échantillons. Nous remarquons aussi qu'avec la taille des échantillons la plus faible (100), l'écart entre les mélanges d'arbres baggés et les mélanges aléatoires d'arbres est le plus faible.

Les bons résultats obtenus avec les mélanges d'arbres baggés reviennent à deux raisons : la première est que les modèles composant le mélange sont optimisés en utilisant l'algorithme MWST, la deuxième cause est l'utilisation des copies bootstrap (perturbation des données d'apprentissage) qui a permis de réduire la variance du modèle final.

La courbe rouge horizontale correspond à la méthode de référence CL. Dans la figure correspondant à 250 et 1000 échantillons, nous remarquons que nos méthodes à base de Bagging surpassent la méthode CL, tandis que les mélanges aléatoires sont moins bons que CL. Dans la figure correspondant à seulement 100 données, les mélanges aléatoires d'arbres sont aussi nettement meilleurs que CL. Ce dernier résultat peut être expliqué par le faite qu'en grande dimension avec

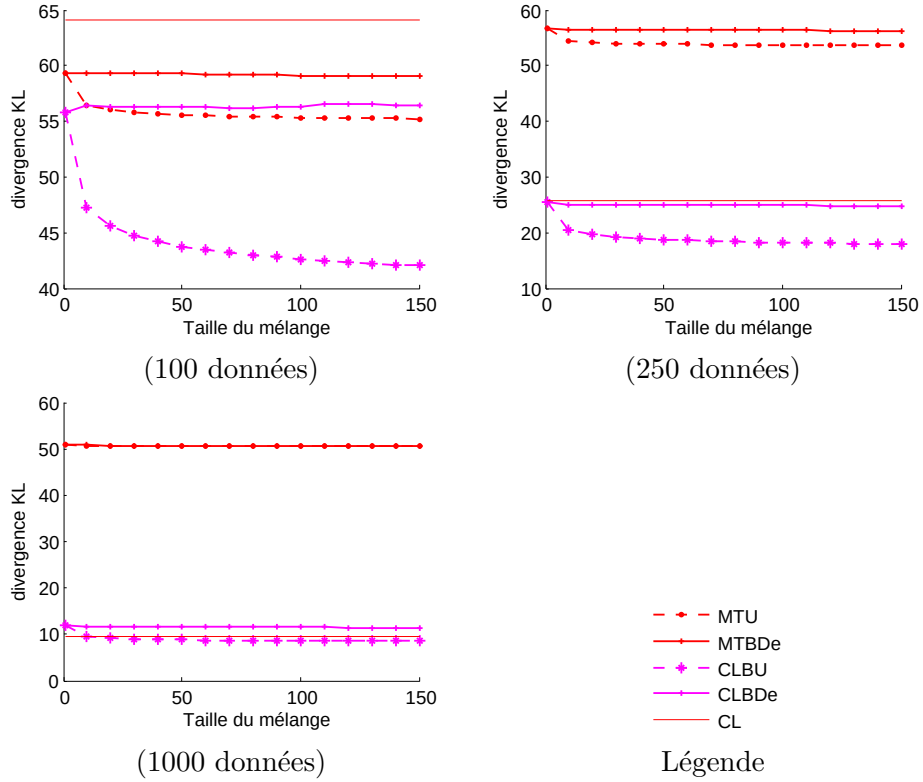


Figure 4.3. Mélanges d'arbres baggés, mélanges aléatoires d'arbres et méthode de base CL ($n = 1000$)

très peu de données, l'optimisation d'un seul modèle risque de faire apparaitre le phénomène de sur-apprentissage. En effet, la mauvaise qualité d'estimation de la méthode CL dans le cas où on ne dispose que de 100 données pour un espace de dimension 1000 revient à ce que cette dernière méthode épouse parfaitement les échantillons disponibles et ensuite converge vers une mauvaise estimation du modèle cible. Ce dernier résultat explique aussi l'intérêt de la randomisation et l'utilisation de modèles faibles dans le cadre où on ne dispose que de très peu de données.

Le succès des méthodes à base de Bagging dans ce cadre de très peu de données explique aussi l'intérêt du principe P&C dans de tels contextes à travers le gain en qualité d'estimation obtenu par la perturbation des données d'apprentissage par Bootstrapping.

4.4.2.2 Influence de la taille des données

De point de vue influence de la taille des données sur la qualité de l'estimation, la figure 4.4 représente les mélanges aléatoires d'arbres MTU et de polyarbres

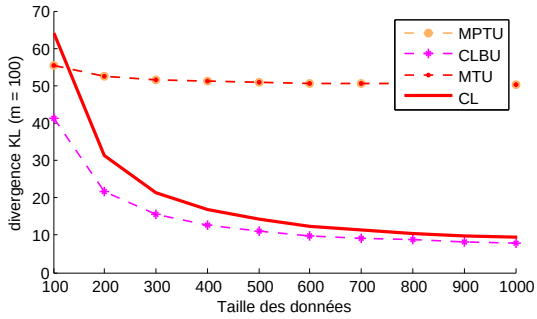


Figure 4.4. Influence de la taille des données sur la qualité de l'estimation des mélanges de taille $m = 100$ ($n = 1000$, $N = 100, ..., 1000$)

$MPTU$, les mélanges d'arbres baggés $CLBU$ et la méthode CL pour différentes tailles de données allant de 100 échantillons et jusqu'à 1000 échantillons. La taille d'un mélange dans cette figure est de $m = 100$.

Nous remarquons d'après cette figure que la qualité de l'estimation de toutes ces méthodes s'améliore avec l'augmentation de la taille de la base de données. Nous remarquons aussi que l'augmentation de la taille des données améliore surtout les méthodes optimisées CL et les mélanges d'arbres baggés $CLBU$. Par contre, les méthodes aléatoires MTU et $MPTU$ gardent une qualité d'estimation similaire (voir légèrement améliorée) pour les différentes tailles des données. Ces derniers résultats reviennent, d'une part, à la nature des algorithmes optimisés qui nécessitent un grand nombre d'individus pour fournir une bonne estimation, sinon, l'estimation peut souffrir du phénomène du sur-apprentissage qui explique surtout le comportement de la méthode CL dans le cas de 100 échantillons (voir le premier point de la figure 4.4 correspondant à 100 échantillons). D'autre part, puisque les méthodes d'échantillonnage aléatoire de modèles MTU et $MPTU$ n'utilisent les données que pour l'apprentissage des paramètres, alors elles sont moins sensibles à la taille des données. Un autre résultat à signaler est que la méthode $CLBU$ est toujours meilleure que les autres méthodes pour les différentes tailles de données.

Le résultat le plus important que nous pouvons tirer de cette discussion est que *la qualité de l'estimation des mélanges aléatoires d'arbres ne dépend pas fortement de la tailles des données et garde un niveau presque constant. Les méthodes utilisant l'algorithme MWST s'améliorent en augmentant la taille des données d'apprentissage.*

Dans le contexte où la taille des données est très faible devant la dimension, toutes nos méthodes sont meilleures que CL.

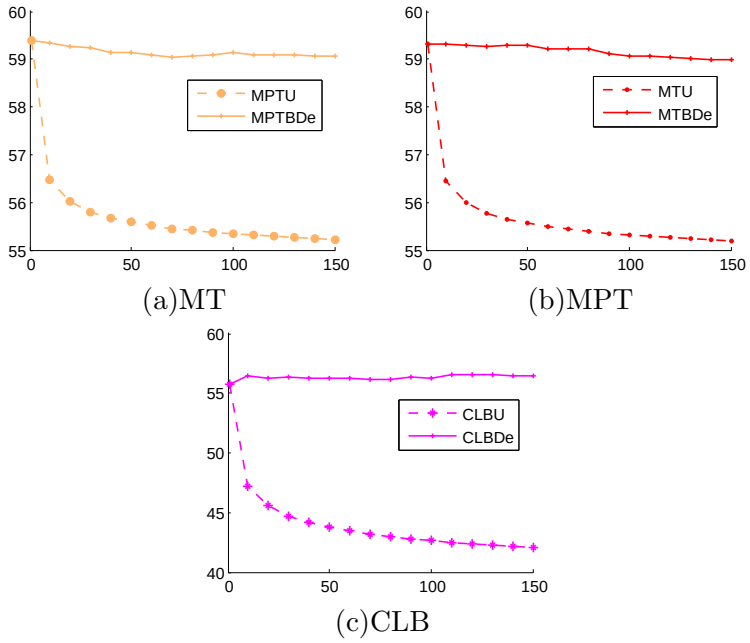


Figure 4.5. Résultats avec des poids uniformes (trait discontinu) Vs. résultats avec poids BDeu (trait continu) (a) mélange de polyarbres, (b) mélange d'arbres, (c) mélange d'arbres baggés ($n = 1000$, $N = 100$)

4.4.2.3 Influence du schéma de pondération

Pour déterminer les poids du mélange, nous avons considéré les deux schémas de pondération présentés dans la section 4.3.3 (uniforme et Bayésien).

La figure 4.5 illustre les résultats obtenus avec les deux schémas de poids pour la construction des mélanges aléatoires d'arbres MT, les mélanges aléatoires de polyarbres MPT et les mélanges d'arbres baggés CLB pour les 10 distributions cibles de type DAG et pour des ensembles de données de taille 100.

D'après cette figure, nous remarquons que *les résultats obtenus avec les poids uniformes sont nettement meilleurs que ceux avec les poids bayésiens. Ce même résultat est aussi observé pour les données de taille supérieure (*250 *et* 1000*).*

Pour étudier d'avantage l'intérêt de l'utilisation des poids bayésiens, nous avons pensé à tester leur utilisation en fonction de la taille des données, et ce pour les deux types de mélanges (mélanges aléatoires d'arbres et mélanges d'arbres baggés). Pour cela nous avons suivi le même protocole décrit dans la section 4.4.2.2 mais en présentant pour chaque méthode la courbe relative aux poids uniformes ainsi que celle relative aux poids BDeu et en utilisant des tailles de données allant jusqu'à 10000 échantillons. Les résultats sont exposés dans la figure 4.6. D'après

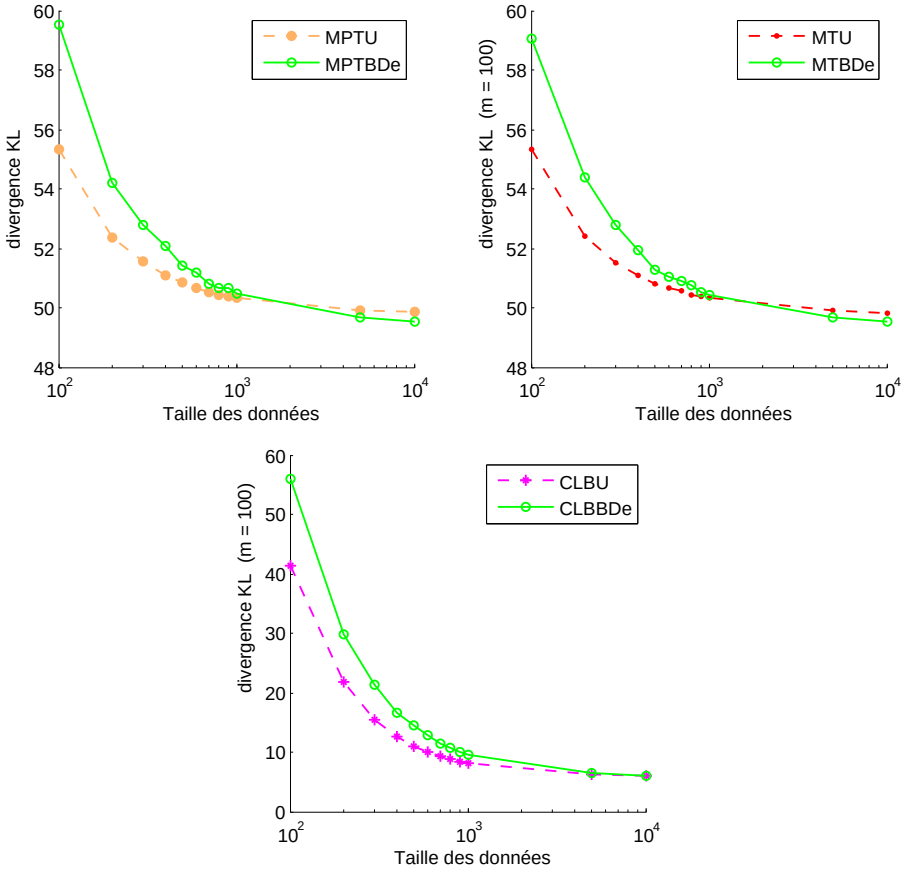


Figure 4.6. Influence du schéma de pondération sur les mélanges à base d'heuristiques de recherche de l'arbre optimal ($n = 1000$, $N = 100, ..., 10000$)

cette figure, nous remarquons que les méthodes utilisant des poids $BDeu$ sont moins bonnes de point de vue qualité d'estimation que les méthodes utilisant des poids uniformes dans le cas où les données sont de taille réduite. Les poids $BDeu$ n'améliorent la qualité d'estimation de nos méthodes que pour des tailles de données élevées (2000 pour MPT et MT, 5000 pour CLB).

Le résultat le plus important qui peut être constaté à partir des résultats obtenus pour l'utilisation des poids bayésiens est : *Les poids bayésiens nécessitent beaucoup de données pour être significatifs.*

4.4.2.4 Influence de la taille du mélange

La façon avec laquelle nous représentons les résultats obtenus pour chacune de nos variantes (qualité de l'estimation en fonction de la taille du mélange) nous permet de voir l'influence de la taille du mélange sur la qualité de l'estimation de nos mélanges. D'après toutes les figures de cette section, nous remarquons que la qualité de l'estimation s'améliore avec l'augmentation de la taille du mélange. Cette amélioration est plus nette (forte pente de la courbe) avec l'ajout des premiers éléments dans le mélange, et se stabilise (faible pente de la courbe) après un certain nombre de composants ajoutées (à peu près 50 modèles dans le mélange).

En conclusion, l'ajout de nouvelles composantes dans le mélange de modèles ne dégrade pas la qualité du mélange même s'il ne l'améliore pas ce qui confirme que nos méthodes ne tendent pas à sur-apprendre.

4.4.2.5 Complexité en temps de calcul

D'après la section 4.4.2.1, nous avons vu que les mélanges d'arbres baggées sont nettement meilleurs que ceux aléatoires de point de vue qualité d'estimation. Toutefois ces méthodes à base du Bagging, dont la complexité est quadratique, sont beaucoup plus coûteuses en terme de complexité algorithmique que les méthodes aléatoires, de complexité linéaire. Puisque nous nous intéressons particulièrement aux espaces de grande dimension, une complexité quadratique en fonction de cette dimension s'avère coûteuse.

Pour compléter cette étude des besoins de nos méthodes, nous avons mesuré le temps de calcul nécessaire pour la construction d'un mélange de taille $m = 150$ pour chacune de ces méthodes. Le tableau 4.4 contient les résultats obtenus, en terme de temps de calcul, pour un ensemble de données de taille 100 et correspondant à un espace de 1000 variables.

Table 4.4. Temps de calcul en secondes ($n = 1000$, $N = 100$)

	1 arbre	150 arbres
MTU	<1	37
CLBU	140	21330

D'après le tableau 4.4, le temps nécessaire pour construire un mélange de 150 arbres optimaux sur des copies Bootstrap des données initiales ($CLBU$) est plus grand que $21330/37 \succ 500$ fois le temps nécessaire pour construire un mélange de même taille d'arbres aléatoirement échantillonnés (MTU). Ce facteur est assez grand et peut s'avérer coûteux si on augmente la dimension du problème.

Ce résultat confirme bien la discussion de point de vue complexité algorithmique.

4.4.2.6 Comparaison par rapport aux méthodes de base

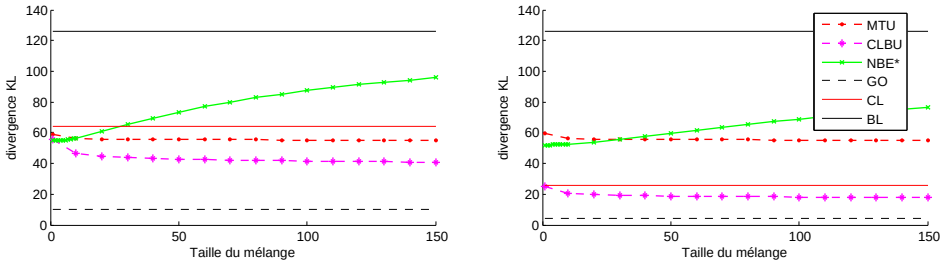
Afin de pousser l'évaluation de nos approches dans le cadre des problèmes de grande dimension et de très faible données, nous avons comparé, dans les mêmes conditions, nos méthodes avec celles de l'état de l'art décrites dans la section 4.4.1.

La figure 4.7 contient les résultats obtenus pour pour les mélanges aléatoires d'arbres MTU, les mélanges d'arbres baggés $CLBU$, la méthode CL, les méthodes de base GO et BL, et la méthode $NBE*$, pour deux tailles de données 100 et 250.Vu le temps de calcul nécessaire pour l'algorithme EM, les résultats de la méthode $MixTree$ pour des distributions de taille $n = 1000$ variables ne sont pas encore prêts. Nous présentons donc des résultats obtenus avec des distributions cibles de taille $n = 8$ variables.

En ce qui concerne la méthode $MixTree$, nous présentons les résultats obtenus avec des distributions cibles de taille $n = 8$ variables. Les résultats avec $n = 1000$ ne sont pas encore prêts vue la grande complexité de cette méthode.

D'après cette figure, toutes nos méthodes sont meilleures que la méthode de base BL et moins bonnes que la méthode GO, et ce pour les deux tailles de données . Ce dernier résultat est logique puisque la méthode GO utilise la vraie structure qui a généré les données d'apprentissage. Le fait que ses paramètres sont appris à partir d'un faible ensemble de données, explique la valeur positive de KL correspondante, qui devrait être nulle si la taille des données est suffisamment grande. La construction d'une telle méthode n'est pas faisable en pratique vue les problèmes de l'apprentissage de structure en grande dimension.

La méthode $NBE*$ obtient des résultats similaires ou légèrement meilleurs



(a) 100 données (b) 250 données

Figure 4.7. Comparaison de nos méthodes (MTU et $CLBU$) par rapport aux méthodes de base et de l'état de l'art ($n = 1000$, $N = 100, 250$, moyennes pour 5 DAG)

que les mélanges aléatoires d'arbres MTU pour un nombre très petit d'éléments dans le mélange. Cependant, contrairement à nos mélanges, cette méthode a tendance à sur-apprendre lorsqu'on ajoute de nouveaux éléments dans le mélange de $NBE*$ (augmenter la cardinalité de la variable classe). Cette tendance au sur-apprentissage (représenté par la remontée de la courbe représentative de $NBE*$) est plus rapide avec la taille des données la plus faible (100 échantillons). Le sur-apprentissage de la méthode $NBE*$ est normal puisque la méthode NBE décrite dans la section 3.3.3.1 comprend une phase d'élagage conçue spécialement pour éviter le sur-apprentissage.

La figure 4.8 représente les résultats obtenus avec nos méthodes (MTU, $CLBU$) ainsi que les méthodes CL, GO, BL, $MixTree$ et $NBE*$ pour un ensemble de 5 distributions cibles de type DAG de taille $n = 8$ noeuds et des ensembles de données de taille $N = 250$. D'après cette figure, nous remarquons que la méthode $MixTree$ fournit des résultats similaires à ceux obtenus avec la méthode $NBE*$, de point de vue qualité d'estimation et comportement. En effet, la qualité d'estimation de $MixTree$ est bonne avec très peu d'éléments dans le mélange, mais se dégrade avec l'augmentation de la taille du mélange et ce à partir d'un faible nombre d'éléments (10 éléments à peu près). Le sur-apprentissage de la méthode $MixTree$ est moins fort que celui de $NBE*$. Ce sur-apprentissage de la méthode $MixTree$ a tendance à disparaitre lorsque le nombre de variables augmente et le nombre d'échantillons diminue, comme nous avons pu l'observer lors d'autres exemples non détaillés dans ce document.

La méthode $MixTree$, quadratique en fonction de n, et utilisant les algorithmes EM et MWST fournit des résultats meilleurs que ceux obtenus avec nos mélanges aléatoires d'arbres MTU, mais ne dépasse pas notre méthode à base de Bagging $CLBU$ qui fournit toujours les meilleurs résultats (à l'exception de la méthode GO

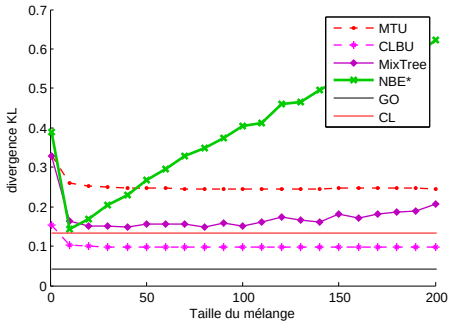


Figure 4.8. Compraison de nos mélanges de modèles simples (MTU et $CLBU$) et la méthode $MixTree$ ($n = 8$, $N = 250$, moyennes pour 5 distributions cibles)

qui n'est pas applicable en pratique).

4.5 Extension au cas continu

Dans ce travail, nous nous sommes restreint au cas de variables discrètes, mais notre approche est bien extensible au cas continu (gaussien). Nous essayerons dans cette section d'expliquer comment nous pouvons créer un mélange de modèles simples pour l'estimation de densité en utilisant des variables continues. Pour cela nous reprenons notre algorithme général et nous discutons l'extension au cas continu de chaque procédure.

Soit $X = \{X_1, \ldots, X_n\}$ un ensemble de n variables. Nous supposons que le jointe de X suit une loi normale multivariée et D un ensemble de données de taille N individus.

Génération aléatoire de structures arborescente

Cette étape est indépendante de la nature des variables et ne fait intervenir que le nombre de variables pour créer une structure arborescente. Cette étape reste alors inchangée (identique au cas discret) en passant au cas gaussien. Seule l'initialisation des paramètres du modèle sera changée, en considérant une densité de distribution gaussienne au lieu d'une table de probabilité (voir section 2.5.3).

Génération de données par un MGP

Nous avons vu dans la section 2.5.5 que l'algorithme 1 *Forward Sampling* permet aussi de générer des échantillons à partir de variables continues.

Apprentissage des paramètres des modèles simples

D'après la section 2.5.3.1, il est possible de faire l'apprentissage de paramètres d'un réseau bayésien gaussien (la distribution jointe est gaussienne multivariée) à partir de données en utilisant les méthodes MV ou MAP.

Calcul des poids

Cette étape ne pose pas de problèmes dans le cas où nous considérons des poids uniformes. Dans le cas d'utilisation de poids BDeu, le score Bayésien, décomposable, d'un graphe G dans le cas de variables continues est donné par la formule 4.4 [Nea04] sous la forme d'un produit des scores bayésiens de chaque variable X_i et ses parents Pa_i (voir chapitre 8 de [Nea04] pour le calcul des valeurs de cette équation).

$$ScoreBD(G, D) = \prod_{i=1}^{n} scoreBD(X_i, Pa_i, D) = \prod_{i=1}^{n} \frac{f(D_{Pa_i \cup \{x_i\}}|G)}{f(D_{Pa_i}|G)} \tag{4.4}$$

Recherche de l'arbre optimal par l'algorithme MWST

L'algorithme MWST contient trois étapes dont deux sont relatives à des manipulations de graphes. La première étape est relative au calcul de la matrice symétrique d'information mutuelle. Cette étape est bien extensible au cas où les variables sont continues. L'expression de l'information mutuelle pour deux variables X_A et X_B continues est donnée par l'équation 4.5.

$$I(X_A, X_B) = \int_{X_B} \int_{X_A} f(X_A, X_B) \log \left(\frac{f(X_A, X_B)}{f_1(X_A) f_2(X_B)} \right) dX_A dX_B \tag{4.5}$$

Comme les lois $f(X_A)$ et $f(X_A, X_B)$ n'ont pas une forme paramétrique simple et connue, cela rend l'estimation de ces lois marginales aussi difficile que l'estimation de la loi jointe en elle même [GL09]. Certains travaux ont proposé d'estimer ces lois par des méthodes non paramétriques. Citons [GL09] qui utilise l'équation 4.6 pour estimer l'information mutuelle de X_A et X_B, dans une gaussienne multivariée, en fonction du coefficient de corrélation ρ_{AB} entre ces deux variables.

$$I(X_A, X_B) = -\log(1 - \rho_{AB}^2) \tag{4.6}$$

[EdAL10] a utilisé cette même estimation dans le cas de gaussiennes multivariées, et a proposé une extension au cas de distributions gaussiennes conditionnelles (mélange de variables discrètes et variables continues).

[GLL+10] et [LLW10] proposent d'utiliser une estimation non paramétrique de type noyau (continu Hölder) pour estimer les lois marginales de X_A et (X_A, X_B) et remplir la matrice d'information mutuelle qui sera ensuite donnée en entrée de l'algorithme de Kruskal pour l'apprentissage de forêts et d'arbres optimaux dans le cas de gaussiennes multivariées.

Une deuxième solution pour apprendre l'arbre optimal à partir de données continues est de remplacer l'information mutuelle par la variation de score comme a été évoqué dans la section 2.8.2. Dans ce cas il suffit de considérer un score décomposable tel que le score BDeu. [EdAL10] a proposé d'utiliser les critères AIC et BIC dans le cas de distributions gaussiennes conditionnelles.

La deuxième étape de l'algorithme MWST est relative à la procédure de Kruskal. Cette étape est indépendante de la nature des variables puisque l'algorithme de Kruskal n'utilise que la matrice d'information mutuelle déjà remplie.

La troisième étape de l'algorithme MWST est relative à l'orientation des arcs de la structure non orientée (résultat de la procédure de Kruskal), en choisissant un noeud racine et en effectuant un parcours en profondeur dans cette structure. Cette étape est alors indépendante de la nature des variables.

Inférence

Il est aussi possible d'effectuer de l'inférence dans les MGP en utilisant l'algorithme de *Messages locaux*, adapté par [Pea88], dans le cas continu.

En conclusion, notre approche à base de mélanges de modèles simples pour l'estimation de densité en utilisant le principe du P&C est applicable dans le cadre de variables continues.

4.6 Conclusions

Nous avons présenté dans ce chapitre deux propositions de mélanges de modèles aléatoires pour l'estimation de densité en appliquant le principe du P&C pour échantillonner aléatoirement des modèles simples de type arbre et polyarbre. Nous avons aussi présenté une troisième proposition de mélanges d'arbres baggés en appliquant le principe du P&C pour perturber les données d'apprentissage en utilisant la technique du Bootstrapping. Dans ce cas de mélanges, les arbres sont appris à partir des copies bootstrap au moyen de l'algorithme MWST. Cette dernière méthode est de complexité quadratique en fonction du temps, alors que

les deux méthodes relatives aux mélanges de modèles aléatoires sont linéaires en fonction du nombre de variables.

Nous avons ensuite présenté un protocole expérimental qui nous a servi à évaluer nos approches et étudier l'intérêt du principe P&C dans le cadre de problèmes de grande dimension et très peu de données.

Nous avons montré aussi lors de ce chapitre que notre approche de mélanges de modèles simples peut être appliquée dans le cas de variables continues.

Le premier résultat obtenu est que les mélanges de polyarbres ne donnent pas de meilleurs résultats que les mélanges d'arbres.

Le deuxième résultat est que les mélanges d'arbres baggés donnent des résultats meilleurs que les mélanges aléatoires mais ils sont de complexité quadratique en fonction de la dimension du problème.

Nous avons vu aussi que toutes nos méthodes s'améliorent de point de vue qualité d'estimation lorsque la taille du mélange augmente, et elles ne tendent pas à sur-apprendre comme le font d'autres méthodes de l'état de l'art.

Un autre résultat important est que les poids $BDeu$ nécessitent beaucoup de données pour pouvoir être significatifs et améliorer la qualité de l'estimation du mélange par rapport aux poids uniformes.

Les mélanges proposés sont alors une solution potentiellement intéressante pour l'estimation de densité dans les espaces creux aussi bien dans le cas discret, comme nous l'avons étudié, que dans le cas continu, comme nous l'avons évoqué dans la section 4.5. En effet, ils permettent d'obtenir une bonne estimation et de faire l'inférence en un temps linéaire (voir section 4.3.4). Les meilleurs résultats sont obtenus avec les mélanges d'arbres baggés qui ont une complexité quadratique en fonction du nombre de variables, ce qui peut s'avérer coûteux dans le cadre de grande dimension. Il sera alors intéressant d'essayer de réduire la complexité de ces derniers mélanges à base du Bagging tout en gardant une qualité d'estimation similaire.

Le chapitre suivant présente des heuristiques de recherche de l'arbre optimal pour essayer de baisser la complexité totale des mélanges à base du Bagging.

Chapitre 5

Heuristiques sous-quadratiques de mélanges d'arbres

5.1 Introduction

Nous avons vu dans le chapitre précédent que nos méthodes de mélanges d'arbres à base de Bagging représentent un bon estimateur de la densité de probabilité en grande dimension et avec très peu de données. Toutefois, ces méthodes sont de complexité élevée par rapport aux mélanges aléatoires d'arbres qui sont seulement linéaires en fonction du nombre de variables. Une complexité quadratique dans un espace de grande dimension peut s'avérer très coûteuse. En effet, l'écart entre $n = 10$ et $n^2 = 100$ n'est pas comme il l'est entre $n = 1000$ et $n^2 = 1000000$.

La phase la plus coûteuse de ces méthodes est celle de recherche de l'arbre optimal à partir des données par l'algorithme de Chow et Liu. Nous proposons dans ce chapitre d'appliquer, encore une fois, le principe du P&C dans ces méthodes pour diminuer la complexité globale en affaiblissant cette étape de recherche.

Nous commençons dans la section 5.2 par analyser la procédure correspondant à la recherche de l'arbre optimal et localiser la partie qui pourra être perturbée. Nous présentons ensuite dans les sections 5.3.1 et 5.3.2 nos propositions pour affaiblir cette procédure de recherche et diminuer sa complexité algorithmique. La section 5.4 sera consacrée à la partie expérimentale et aux résultats.

Le travail décrit dans ce chapitre a fait le sujet d'une première publication scientifique dans la *Conférence Internationale Computational Statistics* en 2010 [ALW10b]. Ce même travail a été ensuite présenté [ALW10a] lors des 5èmes Jour-

nées Francophones sur les Réseaux Bayésiens Une autre publication [ALSW10] dans le *Workshop on Probabilistic Graphical Models* a aussi vu le jour en confrontant notre approche avec celle proposée en parallèle par [SLW10] (voir section 5.3.3).

5.2 Principe général

Nous commençons dans cette partie par analyser la procédure de recherche de l'arbre optimal à partir de données. Cette procédure est relative à l'algorithme de Chow et Liu.

Cette procédure peut être découpée en deux étapes :

1. calculer l'information mutuelle entre chaque paire de variables pour remplir une matrice symétrique MI d'information mutuelle de taille $n \times n$,
2. rechercher l'arbre de recouvrement maximal en utilisant l'algorithme de Kruskal (Algorithme 4, section 2.8.2).

La première étape est quadratique en fonction du nombre de variables ($n(n-1)/2$ termes à calculer) tandis que la deuxième étape est de complexité $E\log(E)$ où E est le nombre d'arcs considérés dans la matrice MI. Si cette matrice MI est entièrement remplie ($E = n(n-1)/2$ termes indépendants), la complexité de la seconde étape est en $n^2\ \log(n)$.

Nous proposons ici d'appliquer encore une fois le principe du P&C en construisant des modèles intermédiaires appris à partir de matrices MI incomplètes. Le nombre K de termes à considérer est alors un paramètre important pour estimer la complexité totale de la procédure.

Le fait d'utiliser une matrice MI incomplète permet, d'une part de réduire la phase de calcul des termes de la matrice en réduisant leur nombre. D'autre part, cela réduit le temps de calcul de la procédure de recherche de l'arbre optimal (algorithme de Kruskal) à partir de cette matrice, puisque la complexité de cette dernière procédure dépend uniquement du nombre de termes considérés (nombre d'arcs). Le résultat de l'algorithme de Kruskal dans ce cas (matrice MI incomplète) peut être un arbre ou une forêt.

Deux questions se posent pour pouvoir accomplir cette procédure : *Combien de termes calculer ?* et *Comment choisir ces termes ?*

Combien de termes considérer ?

Le but de ces heuristiques est de diminuer la complexité totale de la procédure de

Table 5.1. Exemple de mesure de complexité

n	$n\log(n)$	$\log(n\log(n))$	$\frac{\log(n\log(n))}{n\log(n)}$	n^2
1000	6907	$8,84$	$0,0012$	10^6
5000	42586	$10,65$	$0,00025$	$25\ 10^6$
10000	92103	$11,43$	$0,00012$	10^8

recherche de l'arbre optimal à partir de données. Nous avons vu que la complexité de la première étape est égale au nombre de termes considérés K. La deuxième étape de l'algorithme de Chow et Liu est de complexité $K\log(K)$.

Afin d'étudier l'intérêt de l'approche, nous nous sommes concentrés dans ce travail sur des simulations et des résultats obtenus avec une valeur spécifique de ce paramètre $K = n\log(n)$ au lieu de $K = n(n-1)/2$ de l'approche non perturbée.

Si $K = n\log(n)$, alors la complexité de la première étape sera $n\log(n)$. Dans la deuxième étape, nous considèrerons $E = n\log(n)$ arcs pour calculer l'arbre de recouvrement maximal correspondant, et alors la complexité de cette étape est $E\log(E) = n\log(n)\log(n\log(n))$. Cette complexité est nettement inférieure au quadratique et même très proche du quasi-linéaire. D'après le tableau 5.1, le rapport $\log(n\log(n))/n\log(n) \approx 0$. Cela montre que le facteur $\log(n\log(n))$ reste négligeable devant $n\log(n)$. La complexité $n\log(n)\log(n\log(n))$ est alors très proche de la complexité quasi-linéaire $n\log(n)$.

Il est à noter que si $K = n$, la procédure de recherche de l'arbre optimal à partir de n arcs retourne aux meilleurs des cas un arbre, sinon une forêt.

Comment générer ces termes ?

La génération des K termes dans la matrices MI peut être faite d'une façon totalement aléatoire, et selon une loi uniforme. C'est cette méthode aléatoire inspirée du principe du P&C que nous utilisons comme base pour notre première heuristique décrite dans la section 5.3.1.

Nous considérons aussi une deuxième méthode pour remplir partiellement la matrice MI par K termes. Cette deuxième méthode est toujours basée sur une génération aléatoire, mais elle inclut une phase de récupération et d'exploitation du résultat précédent. Cette deuxième méthode est décrite en détail dans la section 5.3.2.

Algorithme 11 Procédure NaiveDrawMarkovTree sous-quadratique naïve

Entrées: n {*nombre de variables*}

$K = \log(n)$
$MI_i = [0]_{n \times n}$,
$D_i = GenSamples(D, i)$,
Pour $k = 1$ to K **Faire**
  Draw new random pair of numbers $(i_1, i_2) \in \{1, ..., n\}^2$,
  $MI_i[i_1, i_2] = ComputeMI(X_{i_1}, X_{i_2})$
  $MI_i[i_2, i_1] = MI_i[i_1, i_2]$
Fin Pour
$T_i = CompKruskal(MI_i)$,
return T_i.

5.3 Variantes

5.3.1 Heuristique naïve : échantillonnage aléatoire d'arcs

Cette première heuristique, appelée *"échantillonnage aléatoire d'arcs"*, consiste à échantillonner un ensemble de taille K termes différents (échantillonnage sans remise) dans la matrice MI selon une loi uniforme. Ces K termes seront ensuite évalués à partir des données d'apprentissage et serviront par la suite pour construire l'arbre optimal correspondant au moyen de l'algorithme de Kruskal. Cette démarche est ensuite répétée pour construire les m éléments du mélange, d'une façon indépendante. Il est alors possible de construire les m composantes du mélange d'une façon parallèle. La figure 5.3.1 décrit le séquencement de la construction des composants du mélange en fonction du temps. Le principe de cette heuristique est décrit par l'algorithme 11.

Dans cet algorithme, à l'itération i de la construction de l'arbre T_i, $\{(i_1, i_2)\}$ représente un ensemble de paires d'indices générés aléatoirement selon une distribution uniforme. Cet ensemble servira à remplir partiellement la matrice MI_i par la procédure *ComputeMI*. La procédure *CompKruskal* prend en paramètres la matrice partiellement remplie MI_i et construit l'arbre de recouvrement maximal correspondant qui sera retourné par l'algorithme.

Dans l'algorithme 11, nous considérons l'étape de la génération des données d'apprentissage désignée par *GenSamples*. Cette procédure est identique à celle de l'algorithme général, donc correspond à une simple copie identiques pour l'ensemble de données initiales ou à une génération d'une copie bootstrap.

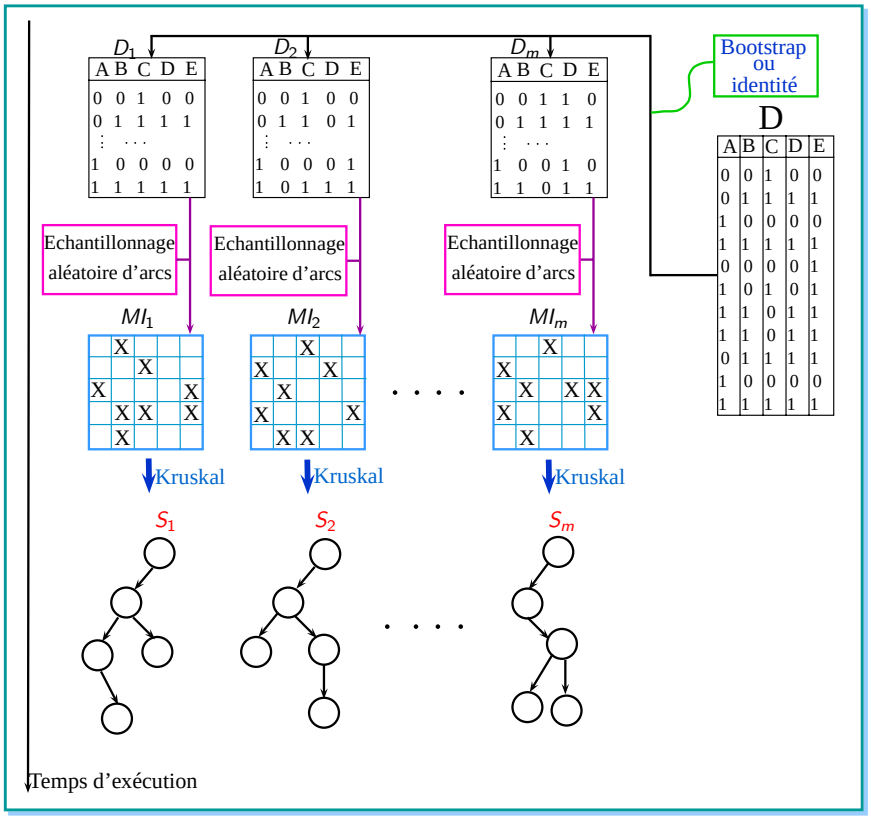


Figure 5.1. Principe du déroulement de l'heuristique naïve en fonction du temps

Cette procédure comprend à la fois un côté aléatoire et un autre optimisé. Le côté aléatoire correspond à l'échantillonnage aléatoire des arcs candidats pour la construction de chaque arbre du mélange. Ce côté aléatoire donne une chance à toutes les connexions entre les variables d'apparaitre même si ces connexions ne sont pas bien représentées par les données observés. Le côté optimisé de cette heuristique revient à choisir parmi les connexions candidates entre les variables celles qui semblent mieux adaptées aux données d'apprentissage sans être sûr que ces connexions formeront une bonne estimation de la densité des données.

De point de vue complexité, la construction d'un mélange de taille m arbres sous-optimaux en utilisant cette heuristique naïve est de :

$$O(m(n\log(n)\log(n\log(n)) + nN)) \propto O(m(n\log(n)\log(n\log(n))))$$

Cette complexité est alors sous-quadratique et très proche du quasi-linéaire.

5.3.2 Heuristique inertielle

Lors de l'exécution de la procédure naïve décrite dans la section précédente, il se peut que certains bons arcs candidats (par rapport aux données) se présentent lors d'une itération et pas lors d'une deuxième, et d'autres se présentent lors de la construction d'un arbre d'une autre itération. Il est donc souhaitable qu'un nombre maximum de bons arcs se présentent lors de la même itération pour produire une estimation proche de l'arbre optimal obtenu en utilisant la totalité de la matrice MI. Dans ce sens, une deuxième idée est considérée pour construire un arbre de recouvrement sous optimal par la procédure *DrawMarkovTree*. Cette idée consiste à tirer avantage du résultat obtenu dans l'itération précédente i pour calculer le prochain résultat à l'itération courante $i+1$. Autrement dit, cette idée consiste à garder et transférer un arc jugé "bon" pour être candidat lors des prochaines sélections. Les indices des arêtes de l'arbre de Markov construit à l'itération i seront alors utilisés en premier pour remplir la matrice MI_{i+1} de l'itération suivante $i+1$, ensuite les K termes de la matrice MI sont complétés aléatoirement. Cette idée est décrite dans l'algorithme 12.

Cette procédure est notée *Inertielle* puisqu'elle rappelle le principe de l'inertie entre une itération et une autre en transférant le résultat obtenu pour mieux optimiser le prochain résultat.

Les procédures *GenSamples*, *ComputeMI* et *CompKruskal* utilisées dans l'algorithme 12 sont les mêmes que dans l'algorithme 11. La procédure *nbEdges*, comme

Algorithme 12 Précédure InertialDrawMarkovTree sous-quadratique inertielle

Entrées: n {*nombre de variables*}

$K = \log(n)$
$MI_i = [\,]_{n \times n}$,
$D_i = GenSamples(D, i)$,
Pour $k = 1$ to $nbEdges(T_{i-1})$ **Faire**
  $(i_1, i_2) = GetIndices(GetEdge(T_{i-1}, k)$,
  $MI_i[i_1, i_2] = ComputeMI(X_{i_1}, X_{i_2})$
  $MI_i[i_2, i_1] = MI_i[i_1, i_2]$
Fin Pour
Pour $k = 1$ to $K - nbEdges(T_{i-1})$ **Faire**
  Draw new random pair of number (i_1, i_2),
  $MI_i[i_1, i_2] = ComputeMI(X_{i_1}, X_{i_2})$
  $MI_i[i_2, i_1] = MI_i[i_1, i_2]$
Fin Pour
$T_i = CompKruskal(MI_i)$,
return T_i.

son nom l'indique, sert à retourner le nombre d'arcs dans une structure qui est égal à $n-1$ dans notre cas (structure d'arbre). La procédure *GetIndices* sert à retourner les indices d'un arc dans un arbre en fournissant les variables correspondantes pour pouvoir repérer le terme de la matrice MI correspondant.

Contrairement à l'heuristique naïve où les éléments du mélange peuvent être construit indépendamment et de façon parallèle, lors de cette heuristique inertielle, la construction de l'élément i dépend fortement de celui de l'étape $i-1$ et ne peut être fait que si la construction de l'élément $i-1$ est terminée. Cette heuristique inertielle est donc séquentielle. La figure 5.2 décrit le séquencement de la construction des composants du mélange en fonction du temps.

Comme l'heuristique naïve, cette heuristique inertielle comprend elle aussi deux côtés, l'un aléatoire et l'autre optimisé. Toutefois, le côté aléatoire de cette heuristique est moins fort puisque le côté optimisé possède un rôle important dans la sélection des termes candidats de la matrice MI. En effet, le fait de garder les bons arcs trouvés à l'itération i et les injecter dans l'itération suivante $i+1$ est une sorte d'optimisation de la sélection des K termes de la matrice MI_{i+1}. Le côté aléatoire ne sert alors plus qu'à échantillonner les K termes de la première itération et ensuite $K-(n-1)$ termes dans les itérations suivantes $2, 3, \ldots, m$.

La complexité de la construction d'un mélange de taille m arbres sous-

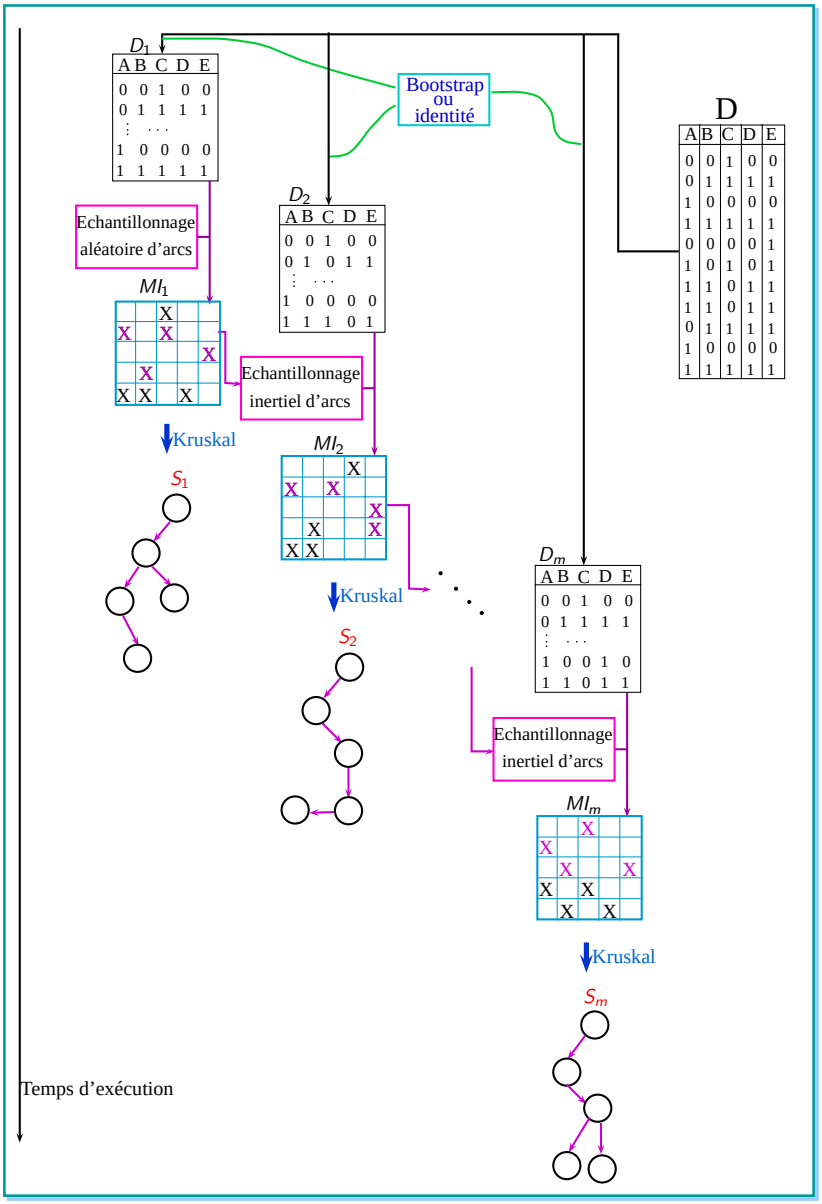


Figure 5.2. Principe du déroulement de l'heuristique inertielle en fonction du temps

optimaux en utilisant cette heuristique inertielle est la même que celle de l'heuristique naïve : complexité sous-quadratique et très proche du quasi-linéaire.

En ce qui concerne l'espace mémoire requis par la construction d'un mélange de m arbres sous-optimaux en utilisant l'une des deux heuristiques naïve et inertielle, d'après la section 4.3.4.2 et puisque ces heuristiques ne nécessitent que $n\log(n)$ termes pour remplir une matrice d'information mutuelle, la complexité en espace est alors $O(n\log(n))$.

5.3.3 Travaux "concurrents"

Comme nos heuristiques proposées précédemment, la méthode décrite dans [SLW10] propose une estimation sous-quadratique de l'arbre optimal à partir de données. Cette idée se base sur l'inégalité triangulaire satisfaite par les mesures de distances : quand un point A est proche d'un point B lui même loin de C, alors A a de fortes chances d'être loin de C aussi. Dans ce contexte, l'information mutuelle est utilisée pour mesurer la "proximité" entre variables. Si les variables composant les deux paires $\{A, B\}$ et $\{A, C\}$ sont proches en terme d'information mutuelle, alors B et C peuvent également être supposées proches.

Cette méthode cherche à éviter le calcul de la totalité de la matrice MI. Pour ce faire, elle construit un partitionnement des variables (appelés clusters) en se basant sur leur information mutuelle, ce qui permet d'identifier les paires de variables qui sont proches dans cette structure et les fournir à l'algorithme MWST de préférence à d'autres. Le principe de cette méthode est décrit par l'algorithme 13.

La complexité algorithmique de cette méthode dépend du nombre d'arcs calculés pour former les clusters qui lui aussi dépend de deux autres paramètres. Cette complexité est au pire des cas quadratique, mais le plus souvent sous-quadratique.

5.4 Validation expérimentale

5.4.1 Protocole expérimental

Pour tester et mesurer la qualité nos deux heuristiques sous-quadratiques, nous avons considéré le même protocole expérimental décrit dans le chapitre 4. En effet, nous avons testé différentes variantes de notre algorithme général en intégrant ces heuristiques de recherche de l'arbre optimal sur le même ensemble de densités cibles (10 DAG de 1000 variables) et sur les mêmes ensembles de données (de taille 100, 250 et 1000) simulés à partir de ces modèles cibles.

Algorithme 13 Algorithme Vertex Clustering ([SLW10])

Entrées: n {*nombre de variables*}

$\mathcal{V} = \mathcal{X}$; $C = \emptyset$; $MI = [0]_{n \times n}$;
Répéter
  $L = GetNewLeader(\mathcal{V}, C)$;
  $C, MI \mathrel{+}= MakeCluster(L, \mathcal{V}, MI_N, MI_C)$.
Jusqu'à $\mathcal{V} = \emptyset$
nbClusters = size(C) ;
Pour $p = 1, \cdots, nbClusters$ **Faire**
  Pour $i_1, i_2 : X_{i_1}, X_{i_2} \in C_p$ **Faire**
    $MI[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
    $MI[i_2, i_1] = MI[i_1, i_2]$.
  Fin Pour
Fin Pour
Pour $\forall q < p : C_q \in Neighbors(C_p)$ **Faire**
  Pour $i_1, i_2 : X_{i_1} \in C_p, X_{i_2} \in C_q$ **Faire**
    $MI[i_1, i_2] = CompMI(X_{i_1}, X_{i_2}, D)$;
    $MI[i_2, i_1] = MI[i_1, i_2]$.
  Fin Pour
Fin Pour
$S = CompKruskal(MI)$;
return S.

Les variantes considérées sont les suivantes (le tableau 5.2 résume ces variantes) :

- Application de l'heuristique "échantillonnage aléatoire d'arcs" sur les données d'apprentissage initiales D, cette variante est notée ESD. Nous avons aussi testé les deux variantes de calcul des poids du mélange. Les mélanges utilisant des poids uniformes seront notés $ESDU$, et ceux utilisant des poids BDeu seront notés $ESDBDe$.
- Application de l'heuristique "échantillonnage aléatoire d'arcs" sur des copies bootstrap B, cette variante est notée ESB. Nous avons aussi testé les deux variantes de calcul des poids du mélange. Les mélanges utilisant des poids uniformes seront notés $ESBU$, et ceux utilisant des poids BDeu seront notés $ESBBDe$.
- Application de l'heuristique inertielle sur les données d'apprentissage initiales D, cette variante est notée $IESD$. Nous avons aussi testé les deux variantes de calcul des poids du mélange. Les mélanges utilisant des poids uniformes seront notés $IESDU$, et ceux utilisant des poids BDeu seront

Table 5.2. Les variantes des heuristiques sous-quadratiques de recherche de l'arbre optimal à partir de données d'apprentissage

Nom	**Génération de structures**	**Données**	**Poids**	**Complexité en temps**
ESBU	NaiveDrawMarkovTree	B	uniformes	$mK\log(K)$
ESBBDe	NaiveDrawMarkovTree	B	BDeu	$mK\log(K)$
ESDU	NaiveDrawMarkovTree	D	uniformes	$mK\log(K)$
ESDBDe	NaiveDrawMarkovTree	D	BDeu	$mK\log(K)$
IESBU	InertialDrawMarkovTree	B	uniformes	$mK\log(K)$
IESBBDe	InertialDrawMarkovTree	B	BDeu	$mK\log(K)$
IESDU	InertialDrawMarkovTree	D	uniformes	$mK\log(K)$
IESDBDe	InertialDrawMarkovTree	D	BDeu	$mK\log(K)$
VCDU	Vert. Clust	D	uniformes	$\leq mn^2\log(n)$
IVCDU	Inertial Vert. Clust	D	uniformes	$\leq mn^2\log(n)$

notés $IESDBDe$.

– Application de l'heuristique inertielle sur des copies Bootstrap B, cette variante est notée $IESB$. Nous avons aussi testé les deux variantes de calcul des poids du mélange. Les mélanges utilisant des poids uniformes seront notés $IESBU$, et ceux utilisant des poids BDeu seront notés $IESBBDe$.

Nous avons aussi intégré dans nos tests la méthode de [SLW10] décrite dans la section 5.3.3 qui propose une estimation sous-quadratique de l'arbre optimal. Cette variante est notée $VCDU$ et utilise l'ensemble de données initiales D et des poids uniformes.

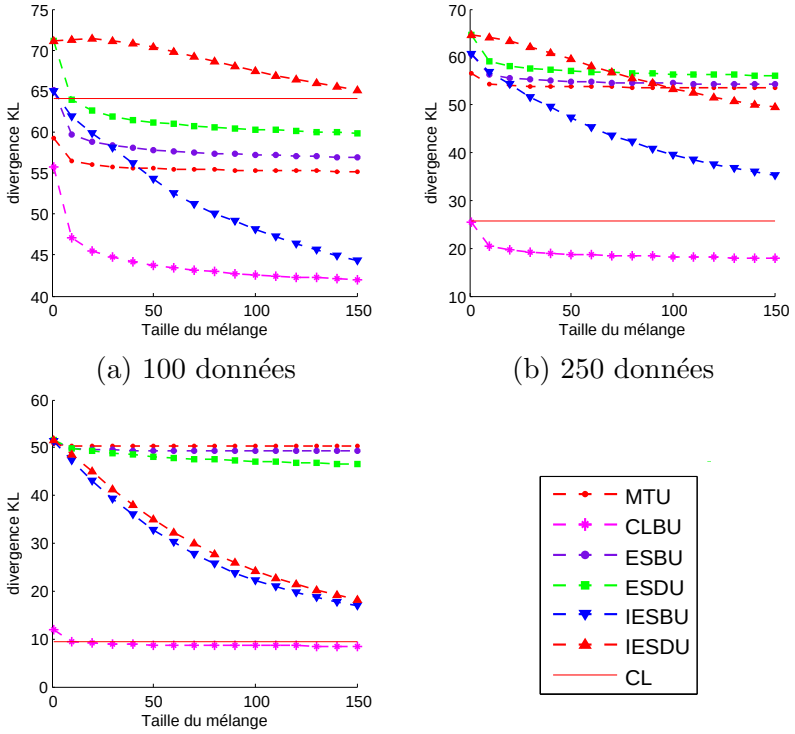
Nous avons aussi considéré une version inertielle de cette dernière méthode (initialement testée dans [ALSW10]), cette variante est notée $IVCDU$ et utilise l'ensemble de données D et des poids uniformes.

5.4.2 Résultats

5.4.2.1 Qualité d'estimation

Heuristique naïve : échantillonnage aléatoire des arcs

La figure 5.3 montre les résultats obtenus avec les deux heuristiques pour les dix distributions cibles de taille 1000 variables pour différentes tailles de l'ensemble de données d'apprentissage. Dans cette figure, nous ne montrons que les résultats obtenus avec des poids uniformes. La courbe en mauve et avec des petits cercles pleins est relative à la méthode $ESBU$ qui échantillonne aléatoirement des arcs candidats pour construire l'arbre de poids maximal à l'aide de l'algorithme MWST



(a) 100 données (b) 250 données

(c) 1000 données Légende

Figure 5.3. Mélanges d'arbres de Markov à base d'heuristiques naïve et inertielle ($n = 1000$)

à partir d'une copie Bootstrap des données initiales. La courbe verte avec des petits carrés est relative à la même méthode mais l'apprentissage de l'arbre optimal se fait sur les données initiales directement.

Cette figure 5.3 montre que ces deux variantes de l'heuristique naïve sont sensiblement meilleures que les mélanges aléatoires d'arbres MTU dans le cas où la taille des ensembles de données est de 1000, mais elles sont moins bonnes que MTU si la taille des données est uniquement de 100 ou 250 échantillons. Ce premier résultat nous montre qu'en général cette heuristique naïve est équivalente aux mélanges d'arbres aléatoires MTU et meilleure que les mélanges d'arbres baggés $CLBU$.

L'utilisation du principe du Bagging pour construire les mélanges d'arbres sous-optimaux confirme le résultat obtenu dans le chapitre précédent : s'il y a peu de données disponibles, le fait d'utiliser des copies Bootstrap est meilleur qu'utiliser les données initiales. Dans ce cadre, l'augmentation du degré d'aléatoire avec le Bagging permet d'avoir de meilleurs résultats avec les données de plus petites tailles. Ce résultat se traduit par le fait que $ESBU$ est meilleur que $ESDU$ pour 100 et 250 données, mais moins bon pour 1000 échantillons.

En conclusion, *si nous disposons de très peu de données (*100 *ou* 250 *échantillons), les mélanges aléatoires d'arbres MTU sont meilleurs que ceux ESBU qui sont à leur tour meilleurs que ESDU. Cet ordre s'inverse si nous disposons d'un ensemble plus important d'échantillons dans la base de données.*

Heuristique inertielle

Dans la figure 5.3, la courbe en bleu et avec des petits triangles pleins dirigés vers le bas est relative à la méthode inertielle $IESBU$ qui utilise une copie Bootstrap pour construire un arbre sous-optimal à chaque itération de l'algorithme. La courbe rouge avec des petits triangles pleins dirigés vers le haut est relative à la méthode inertielle $IESDU$ qui utilise l'ensemble de données initiales pour apprendre l'arbre sous-optimal.

La figure 5.3 montre que la courbe représentative de la méthode $IESBU$ a un comportement différent de celles des autres méthodes. En effet, cette courbe décroit avec une vitesse plus importante que celles des autres méthodes : plus on rajoute des éléments dans le mélange, meilleure est l'estimation du modèle cible. Le même comportement est observé pour la méthode $IESDU$ surtout dans le cas où on dispose de 1000 échantillons.

Le deuxième résultat observé est que la méthode $IESBU$ gagne en qualité

d'estimation par rapport à toutes les autres méthodes, à l'exception des mélanges d'arbres baggés, quand le nombre de composantes dans le mélange augmente. *Cette méthode $IESBU$ tend vers la méthode à base du Bagging et l'approche quand on ne dispose que de* 100 *échantillons.* Quand à la méthode $IESDU$, elle fournit un résultat similaire à celui obtenu avec la méthode $IESBU$ dans le cas où la taille des données est de 1000 échantillons, mais souffre du sur-apprentissage dans le cadre de très peu de données (100 et 250 échantillons), même si la qualité d'estimation du mélange semble s'améliorer avec l'augmentation du nombre d'éléments dans le mélange et cette méthode tend vers CL (voir Figure 5.3 (a)). Ce dernier comportement est expliqué par le fait que les bons arcs qui représentent mieux les données observées D sont découverts par la récupération des bons arcs d'une itération à une autre.

Le mariage du principe du Bagging avec l'optimisation de la recherche des modèles s'avère bien réussi dans le cadre de grande dimension et très peu de données. En effet, comme nous l'avons vu dans la section 5.3.1 pour les variantes utilisant notre heuristique naïve, la qualité d'estimation des mélanges $IESBU$ est nettement meilleure que celle des mélanges $IESDU$ lorsque nous ne disposons que de 100 ou 250 échantillons pour un espace de dimension 1000.

En conclusion, nous avons montré que l'heuristique $IESBU$ est plus intéressante que $IESDU$ tout en approchant les résultats de la meilleure méthode $CLBU$.

5.4.2.2 Influence de la taille des données

Pour faire apparaitre l'influence de la taille des données d'apprentissage sur la qualité d'estimation de nos mélanges d'arbres, nous avons considéré un ensemble d'expériences où l'on mesure KL pour un mélange de modèles de taille fixe $m = 100$ et pour différentes tailles des données $N = 100, 200, \ldots 1000$. Pour chacune des méthodes de mélanges d'arbres sous-optimaux ainsi que les méthodes CL et $CLBU$, ces expériences sont répétées pour cinq distributions cibles de taille $n = 1000$ et pour 10 bases de données différentes pour chaque distribution cible.

la figure 5.4 montre les résultats obtenus. Cette figure montre que la méthode $CLBU$ gagne par rapport à toutes les autres méthodes et pour toutes les tailles des données, et que seule la méthode CL l'approche quand la taille des données est très importante. Nous remarquons aussi que toutes les méthodes à base du bagging ($CLBU$, $IESBU$ et $ESBU$) sont nettement meilleures que la méthode CL lorsque la taille des données est faible (< 3000 pour $CLBU$, < 250 pour

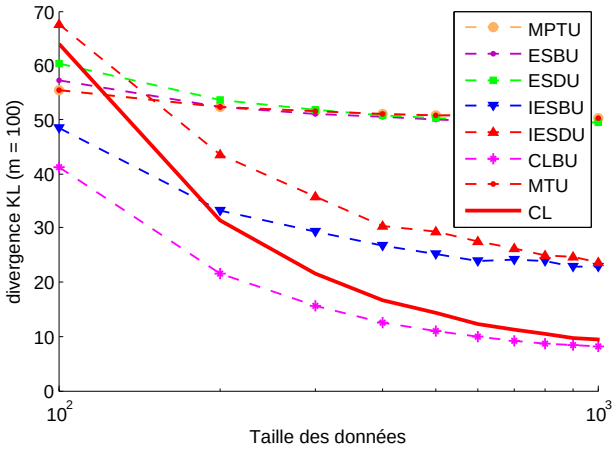


Figure 5.4. Influence de la taille des données sur la qualité de l'estimation des mélanges de taille 100

$IESBU$ et < 120 pour $ESBU$). Même la méthode $ESDU$ semble meilleure que CL pour un ensemble de données de taille 100.

Ces résultats confirment les résultats énoncés précédemment : *le principe du Bagging permet de meilleurs résultats dans le cadre de très peu de données devant la dimension de l'espace.*

5.4.2.3 Influence du schéma de pondération

La figure 5.5 montre des courbes représentatives des résultats obtenus pour chacune des méthodes à base d'heuristiques, en utilisant les deux schémas de pondération (uniforme et $BDeu$), sous la forme d'une courbe moyenne sur cinq distributions cibles 10 bases de données. Cette courbe est relative à un mélange de taille 100 et elle représente la divergence KL en fonction de la taille des données.

D'après la section 4.4.2.3, l'utilisation des poids $BDeu$ pour les mélanges d'arbres nécessite un grand nombre d'échantillons pour être significative et donne de meilleurs résultats que les poids uniformes. Nous avons observé le même comportement de ces poids pour les mélanges à base d'heuristiques naïve et inertielle.

5.4.2.4 Influence de la taille du mélange

Comme nous l'avons vu pour les mélanges aléatoires et les mélanges d'arbres baggés, la qualité d'estimation des mélanges d'arbres sous optimaux obtenus avec l'une de nos deux heuristiques s'améliore quand la taille du mélange augmente. D'après la figure 5.3, nous remarquons que les courbes représentatives des méthodes $IESDU$ et $IESBU$ ont un comportement différent de celui des courbes

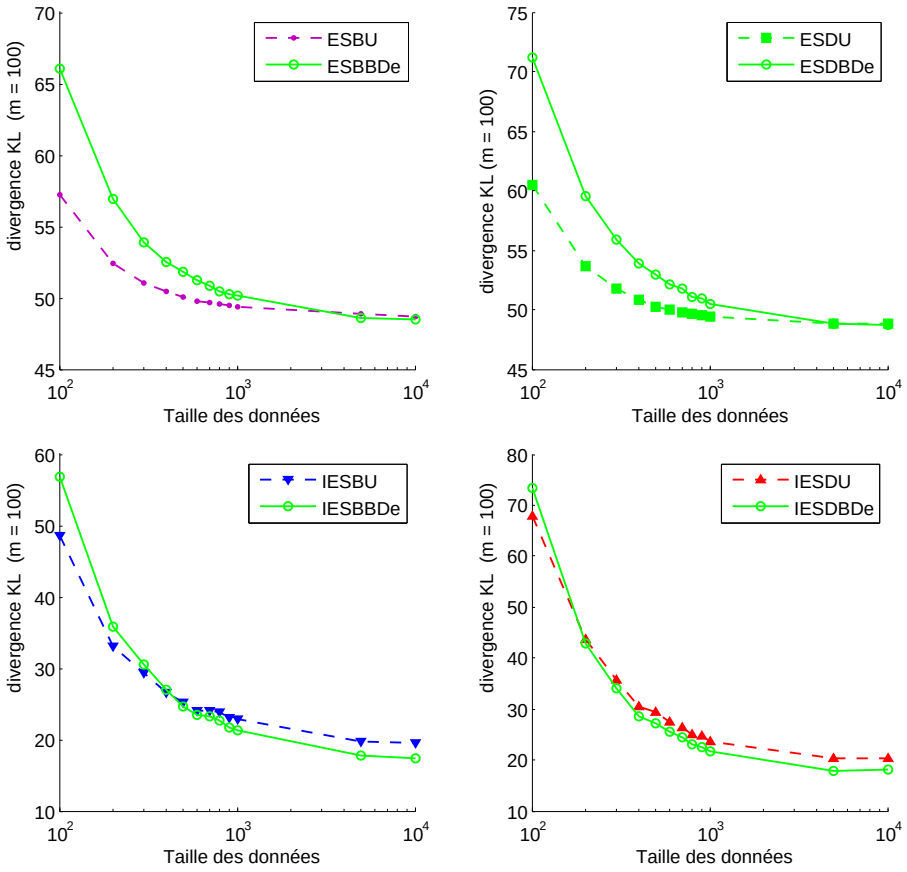


Figure 5.5. Influence du schéma de pondération sur les mélanges à base d'heuristiques de recherche de l'arbre optimal

relatives aux autres méthodes. En effet, ces courbes continuent de décroitre lorsque la taille du mélange augmente et ne semblent pas se stabiliser comme c'est le cas des autres méthodes. Ce comportement est lié à la découverte continue de meilleures relations entre les variables (arcs) qui viennent s'ajouter aux relations déjà découvertes lors des itérations précédentes pour améliorer encore plus la qualité de l'estimation du mélange.

Il est donc intéressant de considérer des mélanges de grandes tailles pour les méthodes inertielles afin d'obtenir une meilleure qualité d'estimation, contrairement aux autres méthodes (MTU, $CLBU$, $ESDU$ et $ESBU$) où un mélange de 50 éléments suffit (voir section 4.4.2.4).

D'après la figure 5.3 (a), relative à un ensemble de données de taille très faible (100 échantillons), nous remarquons que la méthode $IESDU$ a tendance de converger vers la méthode CL et la méthode $IESBU$ a se rapprocher de la méthode $CLBU$ à base du Bagging. Ce comportement est logique, puisque la méthode $IESDU$ apprend des arbres sous optimaux sur les données initiales D et le fait que les bons arcs découverts à une itération sont ensuite gardés pour être candidats lors de l'itération suivante, cela amène donc à découvrir, au bout d'un certain nombre d'itérations, tous les arcs de l'arbre optimal obtenu par la méthode CL. De même pour la méthode $IESBU$, elle recherche les bonnes relations entre les variables en apprenant des arbres sous-optimaux sur des copies Bootstrap des données initiales, ce qui fait tendre l'estimation du mélange vers celui du mélange d'arbres baggés.

Pour mieux observer cette convergence, nous avons repris les simulations pour des mélanges de taille plus grande. La figure 5.6 représente les résultats obtenus pour les méthodes CL et $IESDU$ d'une part, et les méthodes $CLBU$ et $IESBU$ d'autres part. La taille du mélange est élevée à $m = 300$ arbres. Ces résultats confirment bien la tendance de la méthode $IESDU$ vers la méthode CL et celle de la méthode $IESBU$ vers la méthode $CLBU$.

5.4.2.5 Comparaison par rapport à d'autres méthodes

A partir de la figure 5.7, il est clair que la méthode $VCDU$ fournit de meilleurs résultats que nos méthodes d'échantillonnage aléatoire d'arcs ($ESBU$ et $ESDU$). Cette comparaison n'est cependant pas juste, parce que $VCDU$ échantillonne presque 33% des arcs à chaque itération, et $ESDU$ ($ESBU$ aussi) n'échantillonne que 1.4%. Pour fournir une comparaison plus exacte, cette dernière méthode ($ESDU$ dans le tableau 5.2 et la figure 5.7) a été modifiée ($ESDU\%$ dans la

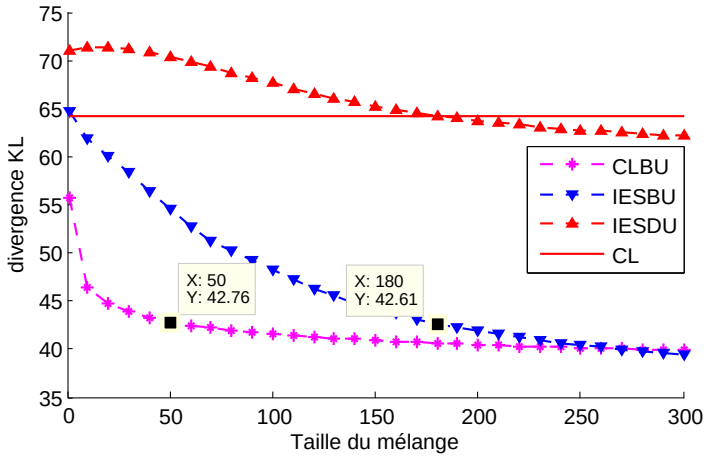


Figure 5.6. Influence de la taille du mélange sur la qualité d'estimation des mélanges d'arbres sous-optimaux. Résultats obtenus pour une seule distribution cible sous la forme de moyenne sur 10 ensembles de données.

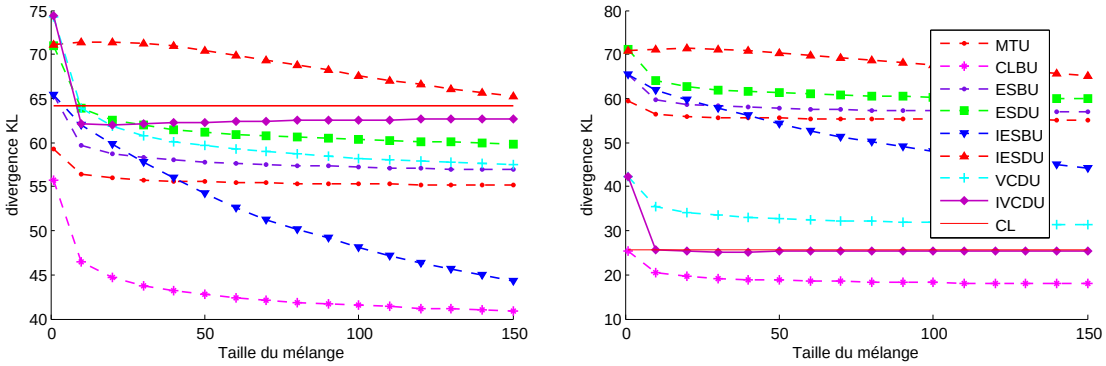
figure 5.8) pour échantillonner un nombre plus élevé d'arcs, soit égal au nombre d'arcs calculés par $VCDU$. Les résultats sont exposés dans la figure 5.8.

Notre heuristique inertielle peut aussi être appliquée à la méthode de partitionnement de variables en clusters. Cette nouvelle variante est notée $IVCDU$. De même que dans le paragraphe précédent, nous avons modifié la variante $IESDU$ en $IESDU\%$ qui échantillonne un nombre d'arcs similaire à celui calculé par $IVCDU$. La figure 5.7 contient les résultats obtenus.

Le premier résultat de cette comparaison est relatif aux deux méthodes $VCDU$ et $ESDU\%$: la méthode de partitionnement des variables en clusters est nettement meilleure que celle utilisant un échantillonnage aléatoire d'arcs utilisant un nombre similaire d'arcs pour construire l'arbre sous optimal. La recherche et la détection des bonnes relations ($VCDU$) est alors plus intéressante que l'échantillonnage aléatoire d'un grand nombre d'arcs.

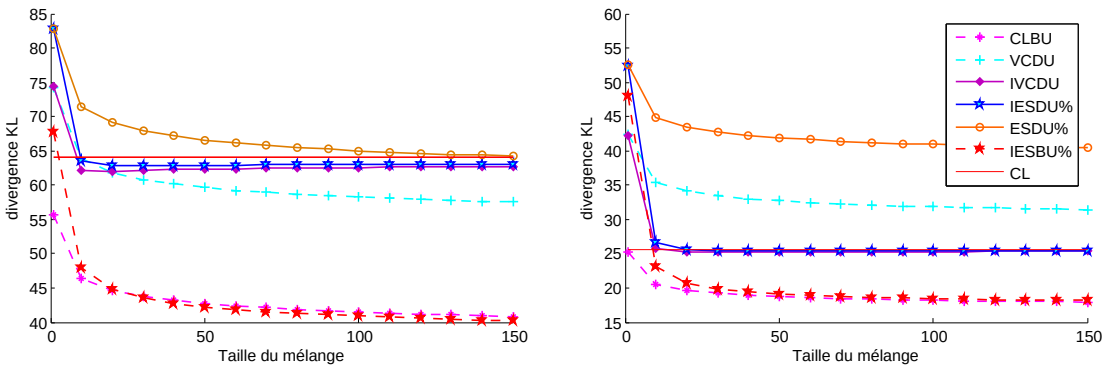
Le deuxième résultat de cette comparaison est relatif aux deux méthodes inertielles $IVCDU$ et $IESDU\%$: en utilisant le principe de report des bons arcs d'une itération à une autre, les deux méthodes inertielles s'avèrent équivalentes de point de vue qualité d'estimation vu qu'elles convergent toutes les deux vers un modèle légèrement meilleur que celui de CL, même si la méthode $IVCDU$ converge plus rapidement.

En conclusion, *notre proposition à base d'un échantillonnage aléatoire d'arcs atteint des résultats similaires à l'heuristique "concurrente" 5.3.3, qui effectue plusieurs tests pour repérer les arcs intéressants.*



(a) 100 données (b) 250 données

Figure 5.7. Comparaison de nos mélanges à bases d'heuristiques et les méthodes de l'état de l'art ($n = 1000$, $N = 100, 250$, moyenne pour 5 DAG)



(a) 100 données (b) 250 données

Figure 5.8. Comparaison de nos mélanges à bases d'heuristiques et les méthodes de l'état de l'art dans les mêmes conditions ($n = 1000$, $N = 100, 250$, moyenne pour 5 DAG)

Table 5.3. Temps de calcul en secondes

	1 arbre	150 arbres
MTU	<1	37
CLBU	140	21330
ESBU	8	1288
ESDU	2.5	507
IESBU	8	1298
IESDU	2.5	567
VCDU	49	9945
IVCDU	49	10870
$ESDU\%$	62	10724
$IESDU\%$	62	10821
$ESBU\%$	67	13566
$IESBU\%$	67	13574

Cette méthode est encore améliorée par la perturbation des données d'apprentissage en utilisant le principe du Bagging. En effet, la méthode $IESBU\%$ correspond à la méthode $IESBU$ mais utilisant un nombre d'arcs similaire que celui utilisé par la méthode $VCDU$. Cette nouvelle variante $IESBU\%$ est nettement meilleure que celle $IVCDU$ et elle permet d'approcher la méthode $CLBU$ quand la taille du mélange augmente (voir figure 5.8).

En conclusion, *nos heuristiques donnent de meilleurs résultats lorsque la valeur de K augmente*. Il sera donc intéressant d'augmenter cette valeur, mais il faut avoir un contrôle sur la complexité et ne pas faire tendre K vers n^2.

5.4.2.6 Complexité en temps de calcul

D'après les sections 5.3.1 et 5.3.2, la complexité des mélanges d'arbres sous-optimaux construit en utilisant l'une de nos deux heuristiques (naïve et inertielle) est sous-quadratique et très proche du quasi-linéaire.

Pour compléter cette étude des besoins de nos méthodes, nous avons mesuré le temps de calcul nécessaire pour la construction d'un mélange de taille $m = 150$ pour chacune de ces méthodes. Le tableau 5.3 contient les résultats obtenus, en terme de temps de calcul, pour un ensemble de données de taille 100 et correspondant à un espace de 1000 variables.

D'après la section 5.3.2, la méthode inertielle $IESBU$ fournit des résultats compétitifs avec ceux de la méthode $CLBU$ et surtout dans le cas où on ne dispose que de très peu de données. Ces résultats se traduisent par le fait que $IESBU$

tend vers $CLBU$ quand le nombre d'éléments dans le mélange augmente.

Du point de vue temps de calcul, ce résultat est aussi confirmé. En effet, dans le cas où on ne dispose que de 100 échantillons dans la base de données d'apprentissage, un mélange $IESBU$ de taille $m = 180$ donne une qualité d'estimation ($KL = 42,61$) très proche de celle obtenue avec un mélange de 50 arbres baggés ($KL = 42,76$) (voir figure 5.6). D'après le tableau 5.3, nous pouvons estimer grossièrement le temps de calcul nécessaire pour construire le premier mélange à environ 1500 secondes. Ce qui est nettement inférieur à celui nécessaire pour construire le deuxième mélange estimé à environ $50 \times 140 \approx 7000$ secondes.

Cette méthode inertielle $IESBU$ de complexité quasi-linéaire s'avère donc une bonne alternative à la méthode quadratique à base du Bagging, qui fournit les meilleurs résultats dans tous les cas.

Les méthodes utilisant un nombre d'arcs similaire (échantillonné aléatoirement pour les méthodes $ESDU\%$ et $IESDU\%$, et détérminé par la procédure de partitionnement des variables pour les méthodes $VCDU$ et $IVCDU$) ont un temps de calcul similaire aussi (voir table 5.3). Ce résultat est logique, puisque toutes ces méthodes calculent le même nombre d'informations mutuelle et ont le même nombre d'arcs candidats à l'entrée de l'algorithme de Kruskal qui permettra de construire l'arbre optimal correspondant.

En ce qui concerne les méthodes $CLBU$ et $IESBU\%$, le temps de calcul correspondant à chacune d'entre elles montre bien que la méthode $IESBU\%$ est préférable puisqu'elle peut être effectuée en un temps largement inférieur à celui de $CLBU$. De plus, $IESBU\%$ a des performances en qualité d'estimation et en rapidité de convergence similaires aux ceux de $CLBU$.

5.5 Conclusion

Nous avons présenté dans ce chapitre deux propositions de mélanges d'arbres sous-optimaux pour l'estimation de densité en appliquant le principe du P&C pour affaiblir la procédure de recherche de l'algorithme de Chow et Liu et baisser la complexité totale de la construction d'un mélange de modèles. Nous avons ensuite utilisé le même protocole expérimental présenté dans le chapitre précédent qui nous a servi pour évaluer nos approches et étudier l'intérêt du principe P&C dans le cadre de problèmes de grande dimension et très peu de données.

Nous avons pu montrer lors de ce chapitre que l'application du principe du P&C sur la procédure de construction des éléments du mélange ainsi que sur les

données d'apprentissage a permis d'atteindre de bons résultats de point de vue qualité d'estimation tout en ayant une complexité algorithmique raisonnable pour être applicable dans le cadre de grande dimension et très peu de données.

Chapitre 6

Conclusion et perspectives

Dans ce chapitre, nous présentons dans la section 6.1 une synthèse des travaux qui ont été abordés dans ce mémoire en récapitulant les principaux résultats et contributions de ce travail. Ensuite, nous évoquons dans la partie 6.2 un ensemble de perspectives.

6.1 Synthèse des travaux présentés dans ce mémoire

Ce paragraphe s'organise autour de trois principales parties : la section 6.1.1 rappelle les motivations des mélanges de modèles graphique probabilistes pour l'estimation de densité en grande dimension, la section 6.1.2 présentera les résultats obtenus avec les mélanges de modèles simples aléatoires ainsi que les mélanges d'arbres à base du Bagging. La section 6.1.3 présentera l'apport de la perturbation de la méthode de Chow et Liu sur la qualité d'estimation de nos mélanges d'arbres sous-optimaux ainsi que sur leur complexité algorithmique.

6.1.1 Motivations de nos approches proposées

L'estimation de densité en grande dimension est face à un ensemble de difficultés. D'une part, l'augmentation de la dimension du problème entraîne une augmentation exponentielle du nombre de paramètres nécessaire pour estimer une densité. De plus, le nombre d'échantillons disponible ne suit pas cette augmentation du nombre de variables et reste relativement très faible, ce qui rend difficile et mauvaise une estimation de densité par les méthodes classiques.

Les modèles graphiques probabilistes fournissent une aide non négligeable pour lutter contre le problème de l'augmentation du nombre de paramètres en fournis-

sant une factorisation de la loi jointe mais souffrent d'un problème de passage à l'échelle.

Le principe du P&C permet de lutter contre les problèmes de passage à l'échelle et de l'estimation à partir de données de très faible taille.

Nous avons alors proposé dans ce travail d'appliquer le principe du P&C pour la construction de mélanges de modèles graphiques probabilistes simples pour faire de l'estimation de densité en grande dimension.

6.1.2 Mélanges de modèles simples pour l'estimation de densité

Nous avons présenté en premier lieu deux propositions de mélanges de modèles aléatoires pour l'estimation de densité en appliquant le principe du P&C pour échantillonner aléatoirement des modèles simples de type arbre et polyarbre. Nous avons aussi présenté une troisième proposition de mélanges d'arbres baggés en appliquant le principe du P&C pour perturber les données d'apprentissage en utilisant la technique du Bootstrapping. Dans ce cas de mélanges, les arbres sont appris à partir des copies bootstrap au moyen de l'algorithme MWST. Cette dernière méthode est quadratique en fonction du nombre de variables, alors que les deux méthodes relatives aux mélanges de modèles aléatoires sont linéaires.

Nous avons montré que les mélanges proposés sont une solution potentiellement intéressante pour l'estimation de densité dans les espaces creux. En effet, ils permettent d'obtenir une bonne estimation et de réaliser des inférences probabilistes en un temps linéaire grâce à l'utilisation de modèles arborescents. Les mélanges de polyarbres sont équivalents aux mélanges d'arbres de point de vue qualité d'estimation. Les meilleurs résultats sont obtenus avec les mélanges d'arbres baggés qui ont une complexité quadratique en fonction du nombre de variables, ce qui peut s'avérer coûteux dans le cadre de grande dimension. Pour cela, nous avons proposé un autre type de mélanges en appliquant encore une fois le principe du P&C pour essayer de baisser la complexité de ces derniers mélanges.

Nous avons aussi montré que nos approches proposées pour l'estimation de densité par les mélanges d'arbres sont extensibles au cas continu.

6.1.3 Heuristique de recherche de l'arbre optimal

Nous avons présenté dans ce travail deux propositions de mélanges d'arbres sous-optimaux pour l'estimation de densité en appliquant le principe du P&C pour affaiblir la procédure de recherche de l'algorithme de Chow et Liu et baisser

la complexité totale de la construction d'un mélange de modèles. Nous avons ensuite utilisé le même protocole expérimental présenté dans le chapitre précédent qui nous a servi pour évaluer nos propositions et étudier l'intérêt du principe P&C dans le cadre de problèmes de grande dimension et très peu de données.

Nous avons pu montrer que nos deux heuristiques (naïve et inertielle) de complexité quasi-linéaire donnent de bons résultats dans le cas où les données disponibles sont de faible taille devant la dimension de l'espace et que notre heuristique inertielle est meilleure que celle naïve. Nous avons montré aussi que le principe du Bagging permet d'améliorer les résultats dans ce même cadre. En effet, notre méthode inertielle utilisant des copies Bootstrap ($IESBU$), quasi-linéaire, s'avère une bonne alternative à la méthode quadratique à base du Bagging ($CLBU$), qui fournit les meilleurs résultats dans tous les cas.

Un autre résultat important est que nos heuristiques donnent de meilleurs résultats lorsque la valeur de K (nombre de termes considéré pour remplir la matrice d'information mutuelle) augmente. Il sera donc intéressant d'étudier l'effet d'autres valeurs de ce paramètre en augmentant sa valeur, mais tout en contrôlant la complexité et ne pas faire tendre K et donc la complexité globale de la méthode vers n^2

6.2 Perspectives

Pour compléter ce travail, l'étape suivante consiste à tester nos approches sur des distributions cibles représentant des problèmes réels. Même si la dimension des réseaux bayésiens (discret) disponibles sont de taille réduite (quelques centaines de noeuds), il est possible d'en construire un autre de taille plus élevée en utilisant des techniques dites de *tilling* [TSBA06].

Une autre piste de travaux est d'améliorer notre procédure d'échantillonnage de polyarbres en imposant qu'elle soit uniforme dans l'espace des représentant des classes d'équivalence de Markov. Cela nous permettra de mieux étudier l'apport des mélanges de polyarbres.

Plusieurs travaux se sont orientés vers l'utilisation de modèles plus faibles que les arbres, les forêts, dans les espaces de grande dimension. Parmi ces travaux nous citons par exemple [GLL+10, EdAL10]. La motivation de ces travaux est le fait que dans les espaces de grande dimension, les données disponibles sont généralement de taille très faible. L'utilisation de modèles faibles (avec le moins d'arcs) est la solution la plus appropriée d'après [WJ08] pour éviter les problèmes liés comme

le sur-apprentissage. Il est donc intéressant de tester les mélanges de forêts pour l'estimation de densité en grande dimension. Dans ce sens, et fidèlement à nos propositions, nous pouvons tester des mélanges de forêts aléatoires, comme nous pouvons tester les mélanges de forêts optimales construites à partir de copie Bootstrap des données initiaux. Cette perspective peut être motivée par la présence de travaux [GLL+10, EdAL10] qui ont proposé des adaptations de l'algorithme de Chow et Liu pour apprendre une forêt optimale, soit par exemple en incluant une phase d'élagage permettant d'enlever des arcs jugés moins pertinents (avec des poids négatifs par exemple).

Suite au succès que nos propositions ont eu dans le cadre discret, et comme nous avons montré que notre approche est extensible au cas de gaussiennes multivariées, la perspective la plus immédiate est de tester nos approches à base de mélanges de modèles simples dans le cas continu ou même le cas conditionnel gaussien (mélange de variables continues et d'autres discrètes). Cette perspective est motivée par les travaux [EdAL10, GL09] qui proposent une extension de l'algorithme de Chow et Liu pour approcher l'arbre optimal représentant une distribution gaussienne multivariée.

Une autre perspective pour ce travail se place dans le cadre des applications en grande dimension qui continuent à avoir un intérêt important actuellement. Cela peut être constaté par les *workshop* récents qui s'intéressent aux problèmes des grandes dimensions ainsi que les solutions possibles. Parmi ces ateliers nous citons le workshop *High Dimensional Problems and Solutions*[1] qui a eu lieu à Paris du 21 au 22 Juin 2010 et le *4th Workshop on High-Dimensional Approximation*[2] qui aura lieu du 26 au 30 Juin 2011 en Allemagne. Il est alors possible d'intégrer nos approches dans les applications réelles qui nécessitent de faire de l'estimation de densité en grande dimension, tels que les applications de classification ou encore les EDA, et tester l'apport de nos approches par rapport aux approches existantes classiquement utilisées.

1. http ://hdps2010.lille.ensam.fr/hdps2010/
2. http ://hda2011.ins.uni-bonn.de/

Bibliographie

[Aka70] H. Akaike. Statistical predictor identification. *Ann. Inst. Statist. Math.*, 22 :203–217, 1970.

[ALDW08a] S. Ammar, Ph. Leray, B. Defoumy, and L. Wehenkel. High-dimensional probability density estimation with randomized ensembles of tree structured bayesian networks. In *Proceedings of the fourth European Workshop on Probabilistic Graphical Models (PGM'08)*, pages 9–16, Hirtshals, Denmark, 2008.

[ALDW08b] S. Ammar, Ph. Leray, B. Defourny, and L. Wehenkel. Density estimation with ensembles of randomized poly-trees. In *The Annual Belgian-Dutch Machine Learning Conference*, pages 31–32, 2008.

[ALDW09a] S. Ammar, Ph. Leray, B. Defoumy, and L. Wehenkel. Pertubation et combinaison d'arbres de markov pour l'estimation de densité. In *Proceedings of Conférence Francophone sur l'Apprentissage Automatique (CAp 2009)*, pages 65–79, Hammamet, Tunisie, 2009.

[ALDW09b] S. Ammar, Ph. Leray, B. Defourny, and L. Wehenkel. Probability density estimation by perturbing and combining tree structured markov networks. In *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)*, pages 156–167, Verona, Italy, 2009.

[Alp04] E. Alpaydin. *Introduction to Machine Learning.* MIT Press, 2004.

[ALSW10] S. Ammar, Ph. Leray, F. Schnitzler, and L. Wehenkel. Sub-quadratic markov tree mixture learning based on randomizations of the chow-liu algorithm. In *the Fifth European Workshop on Probabilistic Graphical Models (PGM-2010)*, pages 17–25, Helsinki, Finland, 2010.

[ALW08] S. Ammar, Ph. Leray, and L. Wehenkel. Estimation de densité par ensembles aléatoires de poly-arbres. In *Proceedings of 4èmes journées*

francophones de réseaux bayésiens JFRB 2008, pages 69–79, Lyon, France, 2008.

[ALW10a] S. Ammar, Ph. Leray, and L. Wehenkel. Mélanges sous-quadratiques d'arbres de markov pour l'estimation de la densité de probabilité. In *Proceedings of 5èmes journées francophones de réseaux bayésiens JFRB 2010*, pages 1–11, Nantes, France, 2010.

[ALW10b] S. Ammar, Ph. Leray, and L. Wehenkel. Sub-quadratic markov tree mixture models for probability density estimation. In *19th International Conference on Computational Statistics (COMPSTAT 2010)*, pages 673–680, Paris, France, 2010.

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete applied mathematics*, 23(1) :11–24, 1989.

[BCG03] Ch . Biernacki, G. Celeux, and G. Govaert. Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate gaussian mixture models. *Comput. Stat. Data Anal.*, 41(3-4) :561–575, 2003.

[Bel57] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Ber09] S. Bernard. *Forêts Aléatoires : de l'Analyse des Mécanismes de Fonctionnement à la Construction Dynamique*. PhD thesis, Université de Rouen, 2009.

[Bes77] J. Besag. Efficiency of pseudolikelihood estimation for simple gaussian fields. *Biometrika*, 64(3) :616–618, 1977.

[BK02] C. Borgelt and R. Kruse. *Graphical Models - Methods for Data Analysis and Mining*. John Wiley & Sons, Chichester, United Kingdom, 2002.

[BMP77] L. Breiman, W. Meisel, and E. Purcell. Variable kernel estimates of multivariate densities. *Technometrics*, 19(2) :135–144, 1977.

[Bre96a] L. Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[Bre96b] L. Breiman. Bias, variance, and arcing classifiers. Technical report, 1996.

[Bre98] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3) :801–824, 1998.

[Bre01] L. Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[CB06] Ch. Bouveyron. *Modélisation et classification des données de grande dimension : application à l'analyse d'images.* PhD thesis, Université Joseph-Fourier - Grenoble I, 09 2006.

[CD85] G. Celeux and J. Diebolt. The sem algorithm : a probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Computational Statistics Quarterly*, 2(1) :73–92, 1985.

[CDLS99] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Statistics for Engineering and Information Science. Springer-Verlag, 1999.

[CG92] C. Celeux and G. Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics and Data Analysis*, 14(3) :315–332, 1992.

[CGH94] D. Chickering, D. Geiger, and D. Heckerman. Learning bayesian networks is NP-hard. Technical report, 1994.

[CGK+02] Jie Cheng, Russell Greiner, Jonathan Kelly, David Bell, and Weiru Liu. Learning Bayesian networks from data : An information-theory based approach. *Artificial Intelligence*, 137(1–2) :43–90, 2002.

[CH92] G. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9 :309–347, 1992.

[CH97] D.M. Chickering and D. Heckerman. Efficient approximations for the marginal likelihood of bayesian networks with hidden variables. *Machine Learning*, 29(2-3) :181–212, 1997.

[Chi03] D.M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3 :507–554, November 2003.

[CL68] C.K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3) :462–467, 1968.

[Col02] M. Collins. Discriminative training methods for hidden markov models : Theory and experiments with perceptron algorithms. In *the*

ACL-02 conference on Empirical methods in natural language processing, pages 1–8, 2002.

[Coo90] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3) :393–405, 1990.

[Cra99] H. Cramèr. *Mathematical methods of statistics*. Princeton : Princeton University Press, 1999.

[CRPP02] T. Choudhury, J. M. Rehg, V. Pavlovic', and A. Pentland. Boosting and structure learning in dynamic bayesian networks for audio-visual speaker detection. In *International conference on pattern recongnition*, pages 789–794, 2002.

[CS96] P. Cheeseman and J. Stutz. Bayesian classification(autoclass) :theory and results, 1996.

[DC93] P. Dagum and R. M. Chavez. Approximating probabilistic inference in bayesian belief networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(3) :246–255, 1993.

[Dec98] R. Dechter. Bucket elimination : a unifying framework for structure-driven inference. *Artificial Intelligence*, 1998.

[DHS01] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons Inc., 2001.

[Die00] Th. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees : Bagging, boosting, and randomization. *Machine Learning*, 40(2) :139–157, 2000.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39 :1–38, 1977.

[Dom00] P. Domingos. Bayesian averaging of classifiers and the overfitting problem. In *IN PROC. 17TH INTERNATIONAL CONF. ON MACHINE LEARNING*, pages 223–230. Morgan Kaufmann, 2000.

[Drt09] M. Drton. Discrete chain graph models. *Bernoulli*, 15(3) :736–753, 2009.

[DS40] W. E. Deming and F. F. Stephan. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *Annals of Mathematical Statistics*, 11(4) :427–444, 1940.

[EdAL10] D. Edwards, G. CG. de Abreu, and R. Labouriau. Selecting high-dimensional mixed graphical models using minimal aic or bic forests. *BMC Bioinformatics*, 11 :18, 2010.

[ET93] B. Efron and R. J. Tibshirani. *An introduction to the Bootstrap.*, volume 57 of *Monographs on Statistics and Applied Probability*. Chapman and Hall, 1993.

[FC90] R. M. Fung and K. Chang. Weighing and integrating evidence for stochastic simulation in bayesian networks. In *UAI '89 : Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, pages 209–220, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co.

[FHT08] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. 9(3) :432–441, 2008.

[FK00] N. Friedman and D. Koller. Being bayesian about network structure. In C. Boutilier and M. Goldszmidt, editors, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 201–210, SF, CA, June 30– July 3 2000. Morgan Kaufmann Publishers.

[Fry90] M. Frydenberg. The chain graph markov property. *Scandinavian Journal of Statistics*, 17(4) :333–353, 1990.

[FS95] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[Gei90] D. Geiger. *Graphoids : a qualitative framework for probabilistic inference*. PhD thesis, Berkeley, CA, USA, 1990.

[Geu01] P. Geurts. Dual perturb and combine algorithm. In *Proc. of the Eighth International Workshop on Artificial Intelligence and Statistics*, pages 196–201, 2001.

[GEW06] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1) :3–42, 2006.

[GH94] D. Geiger and D. Heckerman. Learning gaussian networks. Technical report, Redmond, WA, 1994.

[Gid88] B. Gidas. Consistency of maximum likelihood and pseudo-likelihood estimators for gibbs distributions. In *Stochastic Differential Systems,*

Stochastic Control Theory and Applications. W. Fleming and P.L. Lions, eds. Springer, 1988.

[GL09] J. Goldberger and A. Leshem. A gaussian tree approximation for integer least-squares. In *Neural Information Processing Systems (NIPS)*, pages 638–645, 2009.

[GLL+10] A. Gupta, J. Lafferty, H. Liu, L. Wasserman, and M. Xu. Forest density estimation. In *Info. Th. and Applications (ITA) Workshop*, 2010.

[GRS96] W. Gilks, S. Richardson, and D. Spiegelhalter. Markov chain monte carlo in practice. *Interdisciplinary Statistics. Chapman & Hall*, 1996.

[Hen86] M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *Proceedings of the 2nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, pages 149–163, New York, NY, 1986. Elsevier Science.

[HGC95] D. Heckerman, D. Geiger, and M. Chickering. Learning bayesian networks : The combination of knowledge and statistical data. *Machine Learning*, 20, 1995.

[Hin02] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8) :1771–1800, 2002.

[HO02] F. Huang and Y. Ogata. Generalized pseudo-likelihood estimates for markov random fields on lattice. *Annals of the Institute of Statistical Mathematics*, 54(1) :1–18, March 2002.

[HPBM08] S. M. Hanif, L. Prevost, R. Belaroussi, and M. Milgram. Real-time facial feature localization by combining space displacement neural networks. *Pattern Recognition Letters*, 29(8) :1094–1104, 2008.

[HZ] K. Hwang and B. Zhang. Bayesian model averaging of bayesian network classifiers over multiple node-orders : application to sparse datasets. *Structure learning of Bayesian networks using genetic algorithm.*

[ICR04] J.S. Ide, F.G. Cozman, and F.T. Ramos. Generating random bayesian networks with constraints on induced width. In *Proceedingf of the European Conference on Artififcial Intelligence (ECAI 2004)*, pages 323–327, 2004.

[JGJS98] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. In Michael I. Jordan, editor, *Learning in Graphical Models*. Kluwer Academic Publishers, Boston, 1998.

[JLO90] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4 :269–282., 1990.

[JPR08] Y. Jing, V. Pavlovic, and J. M. Rehg. Boosted bayesian network classifiers. *Machine Learning*, 72(2) :155–184, 2008.

[KA09] Z. Kebaili and A. Aussem. A novel hybrid bayesian network structure learning algorithm based on correlated itemset mining techniques. *International Journal of Computational Intelligence Research*, 5(1) :16–21, 2009.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models : Principles and Techniques*. MIT Press (MA), 2009.

[Kir08] S. Kirshner. Learning with tree-averaged densities and distributions. 2008.

[KK09] M. P. Kumar and D. Koller. Learning a small mixture of trees. In *Neural Information Processing Systems (NIPS 2009)*, 2009.

[KL51] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1) :79–86, 1951.

[KMT96] P. Kontkanen, P. Myllymaki, and H. Tirri. Constructing bayesian finite mixture models by the EM algorithm. Technical report, ESPRIT Working Group on Neural and Computational Learning (NeuroCOLT, 1996.

[Kru56] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1) :48–50, February 1956.

[Lau92] S. Lauritzen. Propagation of probabilistics, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87 :1098–1108, 1992.

[LD05] D. Lowd and P. Domingos. Naive bayes models for probability estimation. In *the 22nd international conference on Machine Learning (ICML '05)*, pages 529–536. ACM, 2005.

[Ler92] B. G. Leroux. Consistent estimation of a mixing distribution. *Annals of Statistics*, 20(3) :1350–1360, 1992.

[LLW10] H. Liu, J. Lafferty, and L. Wasserman. Tree density estimation. 2010.

[LQ65] D. O. Loftsgaarden and C. P. Quesenberry. A nonparametric estimate of a multivariate density function. *Ann. Math. Statist.*, 36(3) :1049–1051, 1965.

[LR01] S. Lauritzen and T. S. Richardson. Chain graph models and their causal interpretations. *B*, 64 :321–361, 2001.

[LRS04] E. Lamma, F. Riguzzi, and S. Storari. Exploiting association and correlation rules - parameters for improving the k2 algorithm. In *The 16th European Conference on Artificial Intelligence, ECAI'2004*, pages 500–504. IOS Press, 2004.

[LS88] S. Lauritzen and D. Speigelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Royal statistical Society B*, 50 :157–224, 1988.

[LW89a] S. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17 :31–57, 1989.

[LW89b] S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17(1) :31–57, 1989.

[LZLT09] Y. Lin, S. Zhu, D. Lee, and B. Taskar. Learning sparse markov network structure via ensemble-of-trees models. In *The Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTAT'09)*, volume 5, pages 360–367, 2009.

[MC98] S. Monti and G. F. Cooper. Learning hybrid bayesian network from data. In Michael I. Jordan, editor, *Learning in Graphical Models*. Kluwer Academic Publishers, Boston, 1998.

[MP99] M. Meila-Predoviciu. *Learning with Mixtures of Trees*. PhD thesis, MIT, 1999.

[MP00] G. McLachlan and D. Peel. *Finite mixture models*. Wiley, 2000.

[MPDW06] A. Mccallum, Ch. Pal, G. Druck, and X. Wang. Multi-conditional learning : generative/discriminative training for clustering and classification. In *AAAI*, 2006.

[MR94] D. Madigan and A.E. Raftery. Model selection and accounting for model uncertainty in graphical models using occam's window. *The American Statistical Association*, 89 :1535–1546, 1994.

[MSA+06] K. Mekhnacha, L. Smail, J. M. Ahuactzin, P. Bessière, and E. Mazer. A unifying framework for exact and approximate Bayesian inference. Technical report, 2006.

[MSL10] R. Mourad, C. Sinoquet, and Ph. Leray. Learning hierarchical bayesian networks for genome-wide association studies. In *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT 2010)*, pages 549–556, 2010.

[Mur98] K. P. Murphy. Inference and learning in hybrid bayesian networks. Technical report, 1998.

[Mur03] K. P. Murphy. Fitting a conditional linear gaussian distribution. Technical report, 2003.

[MY95] D. Madigan and J. York. Bayesian graphical models for discrete data. *International Statistical Review*, 63 :215–232, 1995.

[Nea04] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2004.

[NWL+04] P. Naïm, P-H. Wuillemin, Ph. Leray, O. Pourret, and A. Becker. *Réseaux bayésiens*. Eyrolles, Paris, 2 edition, 2004.

[NWL+07] P. Naïm, P-H. Wuillemin, Ph. Leray, O. Pourret, and A. Becker. *Réseaux bayésiens*. Eyrolles, Paris, 3 edition, 2007.

[OOM04] M. Ouerd, B. J. Oommen, and S. Matwin. A formal approach to using data distributions for building causal polytree structures. *Information Sciences - Informatics and Computer Science*, 168(1-4) :111–132, 2004.

[Pea94] K. Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London A*, (186) :343–414, 1894.

[Pea85] J. Pearl. Bayesian networks : a model of self-activated memory for evidential reasoning. In *Technical Report 850021 (R-43), UCLA Computer Science Department Technical Report, and in Cognitive Science Society, UC Irvine*, 1985.

[Pea87] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32(2) :245–257, 1987.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[Pea91] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, volume second edition. Morgan Kaufmann, 1991.

[PP85] J. Pearl and A. Paz. Graphoids : A graph based logic for reasonning about relevance relations. In *Technical Report 850038 (R-53-L), Cognitive Systems Laboratory, University of California, Los Angeles.*, 1985.

[PV91] J. Pearl and T. S. Verma. A theory of inferred causation. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR'91 : Principles of Knowledge Representation and Reasoning*, pages 441–452, San Mateo, California, 1991. Morgan Kaufmann.

[Qui89] A. Quiroz. Fast random generation of binary, t-ary and other types of trees. *Journal of Classification*, 6(1) :223–231, December 1989. available at http ://ideas.repec.org/a/spr/jclass/v6y1989i1p223-231.html.

[Rid02] G. Ridgeway. Looking for lumps : boosting and bagging for density estimation. *Computational Statistics & Data Analysis*, 38(4) :379 – 392, 2002.

[Rob77] R. W. Robinson. Counting unlabeled acyclic digraphs. In C. H. C. Little, editor, *Combinatorial Mathematics V*, volume 622 of *Lecture Notes in Mathematics*, pages 28–43, Berlin, 1977. Springer.

[Ros56] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3) :832–837, 1956.

[RP88] G. Rebane and J. Pearl. The recovery of causal poly-trees from statistical data. *Int. J. Approx. Reasoning*, 2(3) :341, 1988.

[RS61] H. Raïffa and R. Schlaifer. *Applied statistical decision theory.* Division of Research, Graduate School of Business Adminitration, Harvard University, 1961.

[RS02] S. Rosset and E. Segal. Boosting density estimation. In *Proceedings of the 16th International Conference on Neural Information Processing Systems (NIPS)*, pages 267–281, Vancouver, British Columbia, Canada, 2002.

[RW97] K. Roeder and L. Wasserman. Practical bayesian density estimation using mixtures of normals. *Journal of the American Statistical Association*, 92 :894–902, 1997.

[Sch78] G. Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2) :461–464, 1978.

[Sch90] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2) :197–227, 1990.

[SGS93] P. Spirtes, C. Glymour, and R. Scheines. *Causation, prediction, and search*. Springer-Verlag, 1993.

[SGS00] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, 2 edition, 2000.

[Sil86] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, April 1986.

[SLW10] F. Schnitzler, Ph. Leray, and L. Wehenkel. Towards sub-quadratic learning of probability density models in the form of mixtures of trees. In *18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2010)*, pages 219–224, Bruges, Belgium, 2010.

[SP90] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *UAI '89 : Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, pages 221–234, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co.

[Sre01] N. Srebro. Maximum likelihood bounded tree-width markov networks. In *Artificial Intelligence*, pages 504–511, 2001.

[Sri01] R. Srinivasan. *Importance Sampling : Applications in Communications and Detection*. Springer, 2001.

[ST83] D. Scott and J. Thompson. Probability density estimation in higher dimensions. In *Fifteenth Symposium in the Interface*, pages 173–179, 1983.

[SV93] M. Singh and M. Valtorta. An algorithm for the construction of bayesian network structures from data. In *The Ninth Annual Conference on Uncertainty in Artificial Intelligence UAI 93*, pages 259–265. Morgan Kaufmann, 1993.

[TBA06] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1) :31–78, 2006.

[TMC+97] B. Thiesson, Ch. Meek, D. Chickering, D. M. Chickering, and D. Heckerman. Learning mixtures of dag models. In *In Proc. of the Conf. on Uncertainty in AI*, pages 504–513. Morgan Kaufmann, Inc, 1997.

[TMCH97] Bo Thiesson, Christopher Meek, David Maxwell Chickering, and David Heckerman. Learning mixtures of bayesian networks. In *in Cooper & Moral*, 1997.

[TSBA06] I. Tsamardinos, A. Statnikov, L. E. Brown, and C. F. Aliferis. Generating realistic large bayesian networks by tiling. In *In The 19th International FLAIRS Conference*, 2006.

[TSE02] F. Thollard, M. Sebban, and Ph. Ezequel. Boosting density function estimators. In *Machine Learning : ECML 2002*, pages 147–153. Springer Berlin / Heidelberg, 2002.

[TWO+03] Y. W. Teh, M. Welling, S. Osindero, G. E. Hinton, T. Lee, J. Cardoso, E. Oja, and Sh. Amari. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4 :1235–1260, 2003.

[VP91] T. Verma and J. Pearl. Equivalence and synthesis of causal models. In M. Henrion, R. Shachter, L. Kanal, and J. Lemmer, editors, *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 220–227, San Francisco, 1991. Morgan Kaufmann.

[WJ08] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. 1(1–2) :1–5, 2008.

[Wol92] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5 :241–259, 1992.

[Wu83] C. F. Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1) :95–103, 1983.

[ZP94] N. Zhang and D. Poole. A simple approach to bayesian network computations. In *the tenth Canadian Conference on Artificial Intelligence*, 1994.

[ZY08] X. Zhu and Y. Yang. A lazy bagging approach to classification. *Pattern Recogn.*, 41(10) :2980–2992, 2008.

Annexe A

Notions de graphe

Cette annexe introduit certaines notions en relation avec les graphes et elle est inspirée de [BK02].

Définition 8 (*Graphe*)

Un **graphe** *est une paire $G = (V, E)$, où V est un ensemble fini de* **noeuds** *et $E \subseteq V \times V$ est un ensemble fini d'arcs. Un graphe ne contient pas de boucles, c'est-à-dire, il n'existe pas d'arc (A, A) pour tout $A \in V$.*
G est dit **non dirigé** *si*

$$\forall A, B \in V : \quad (A, B) \in E \quad \Rightarrow \quad (B, A) \in E$$

Les deux paires (A, B) et (B, A) sont identifiées et représentent un seul arc (non dirigé), dit arête.
G est dit **dirigé** *si*

$$\forall A, B \in V : \quad (A, B) \in E \quad \Rightarrow \quad (B, A) \notin E$$

Un arc (A, B) est considéré dirigé de A vers B.

Graphes non orientés (non dirigés)

Définition 9 (*Voisinage*)

Soit $G = (V, E)$ un graphe non dirigé. Un noeud $B \in V$ est appelé **adjacent à** *un noeud $A \in V$ ou* ***voisin*** *de A* si et seulement si *il existe une arête entre*

eux, i.e. si et seulement si $(A, B) \in E$. *L'ensemble de tous les voisins de* A,

$$\mathbb{V}(A) = \{B \in V | (A, B) \in E\},$$

est appelé ***voisinage*** *du noeud* A *et l'ensemble*

$$fermeture(A) = \mathbb{V}(A) \cup \{A\}$$

est appelé la ***fermeture*** *de* A.

Définition 10 (*Chaîne, Cycle*)

Une ***chaîne*** *est une suite d'arêtes dans laquelle chaque arête est reliée à la précédente et à la suivante par une extrémité commune. Une chaîne peut être définie par la suite des sommets qu'elle rencontre.*

Un ***cycle*** *est une chaîne finie de longueur supérieure à 2 pour laquelle ses sommets initiaux et finaux sont identiques.*

Définition 11 (*Sous-graphe complet, Clique*)

Soit $G = (V, E)$ *un graphe non dirigé. Un graphe non dirigé* $G_X = (X, E_X)$ *est appelé* ***sous-graphe*** *de* G *(induit par* X*)* si et seulement si $X \subseteq V$ *et* $E_X = (X \times X) \cap E$, *i.e.* si et seulement si *il contient un sous ensemble de noeuds dans* G *et tous ses arrêtes correspondantes.*

Un graphe non dirigé $G = (V, E)$ *est appelé* ***complet*** si et seulement si *son ensemble d'arrêtes est complet, i.e.* si et seulement si *toutes les arrêtes possibles sont présentes, ou formellement* si et seulement si

$$E = V \times V - \{(A, A) | A \in V\}$$

Un sous-graphe complet est appelé ***clique****.*

Graphes orientés (dirigés)

Définition 12 (*Chemin, Circuit*)

Pour un graphe dirigé $\vec{G} = (V, \vec{E})$, *on appelle* ***chemin*** *une suite* $(u_1, \ldots, u_k)$ *d'arcs telle que l'extrémité de chaque arc coïncide avec l'origine du suivant.*

Un ***circuit*** *est un chemin pour lequel le sommet initial et le sommet ter-*

minal sont identiques.

L'entier naturel non nul k est alors appelé la longueur du chemin. Un chemin peut également être identifié par la suite des noeuds $(X_1, \ldots, X_{k+1})$ par lesquels il passe.

Définition 13 (*Squelette*)

Soit $\vec{G}\ (V, \vec{E})$ un graphe dirigé. Soit l'ensemble $E* = \{$arêtes $A, B | (A, B) \in E$ et/ou $(B, A) \in E\}$.

Le graphe non dirigé $G = (V, E*)$ est appelé* ***squelette*** *de G.*

Définition 14 (*Ordre topologique*)

soit $\vec{G}\ (V, \vec{E})$ un graphe dirigé sans circuits. Une numérotation des arcs de $\vec{G}$, i.e. une fonction $o : V \to \{1, \ldots, |V|\}$, verifiant

$$\forall A, B \in V : (A, B) \in \vec{E} \Rightarrow o(A) < o(B)$$

est appelé ***ordre topologique*** *des noeuds de $\vec{G}$.*

Annexe B

Représentation graphique d'indépendance conditionnelle

Cette annexe est relative aux notions de représentation graphique des indépendances onditionnelles et elle est inspirée de [BK02] et [NWL+04].

B.1 Axiomes des modèles d'indépendance

Définition 15 (*Modèles d'indépendance*)

Soit V un ensemble fini et non vide de variables, on note $T(V)$ l'ensemble des triplets $(X \perp Y|Z)$de sous-parties disjointes X, Y, Z de V où X et Y sont non vides. Toute sous-partie de $T(V)$ est un modèle d'indépendance.

$(X \perp Y|Z)_M$ est la proposition logique qui indique que le triplet $(X \perp Y|Z)$ appartient au modèle d'indépendance M.

Définition 16 (*Semi-Graphoïde*)

Un modèle d'indépendance M est un semi-graphoïde s'il satisfait pour tout X, Y, Z, W sous-ensembles disjoints de V :

- *Indépendance triviale :* $(X \perp \emptyset|Z)_M$
- *Symétrie :* $(X \perp Y|Z)_M \Rightarrow (Y \perp X|Z)_M$
- *Décomposition :* $(W \cup X \perp Y|Z)_M \Rightarrow (W \perp Y|Z)_M \wedge (X \perp Y|Z)_M$
- *Union faible :* $(W \cup X \perp Y|Z)_M \Rightarrow (X \perp Y|Z \cup W)_M$
- *Contraction :* $(X \perp Y|Z \cup W)_M \wedge (W \perp Y|Z)_M \Rightarrow (W \cup X \perp Y|Z)_M$

Une relation ternaire $(.\perp.|.)$ qui satisfait les axiomes du semi-graphoïde pour tout X, Y, Z et W est appelée semi-graphoïde.

On peut définir pour chaque modèle d'indépendance $M \subset T(V)$ un semi-graphoïde $SG(M)$ qui est le semi-graphoïde minimal (au sens de l'inclusion) contenant M. $SG(M)$ est la *fermeture* de semi-graphoïde M.

Définition 17 (*Graphoïde*)

Un modèle d'indépendance M est un graphoïde s'il est semi-graphoïde et s'il satisfait :

- *Intersection :* $(W \perp Y|Z \cup X) \wedge (X \perp Y|Z \cup W) \Rightarrow (W \cup X \perp Y|Z)$

Une relation ternaire $(.\perp.|.)$ qui satisfait les axiomes du graphoïde pour tout X, Y, Z et W est appelée graphoïde.

De même que pour le semi-graphoïde, $G(M)$ est la *fermeture* de graphoïde de M.

B.2 Modèles d'indépendance et loi de probabilité

Toute loi de probabilité $\mathcal{P}$ sur un ensemble V de variables définit un modèle d'indépendance. Il suffit de lister l'ensemble des triplets (X, Y, Z) de sous-ensembles disjoints de V qui vérifient la propriété d'indépendance conditionnelle $X \perp Y|Z$. En notant $M_{\mathcal{P}}$ le modèle d'indépendance ainsi crée, on a :

Théorème 1

- $M_{\mathcal{P}}$ possède une structure de semi-graphoïde.
*- **Si** $\mathcal{P}$ est positive **alors** $M_{\mathcal{P}}$ possède une structure de graphoïde.*

Le théorème 1 implique que le modèle d'indépendance issu d'une loi de probabilité a une structure de semi-graphoïde.

Théorème 2

La fermeture F de sous-graphoïde de tout couple d'éléments de $T(V)$ est un modèle d'indépendance conditionnelle probabiliste.
C'est-à-dire : il existe une loi de probabilité P sur l'ensemble des variables V telle que $P(X|Y,Z) = P(X|Z) \Leftrightarrow (X \perp Y|Z)_F$.

La notion de graphoïde introduite par [PP85] et [Gei90] permet de faire le lien entre la notion d'indépendance conditionnelle entre les variables aléatoires, et les notions de *séparation* dans les graphes non orientés et de *d-séparation* dans les graphes orientés. Cette notion justifie alors l'identification entre variables aléatoires et noeuds d'un graphe.

B.3 Modèles d'indépendance et séparation dans les graphes

Les modèles d'indépendance permettent de décrire certaines propriétés en théorie des graphes. Réciproquement, représenter un modèle par un graphe permettrait de visualiser beaucoup plus facilement la relation représentée par ce modèle.

Un graphe est une relation binaire entre les éléments d'un ensemble alors qu'un modèle d'indépendance est une relation ternaire entre sous-parties de cet ensemble. Le lien entre ces deux types de relation est apporté par la notion de *séparation* définie par la définition 18.

Définition 18 (*Séparation*)

Soit un graphe $G = (V, E)$, pour tout triplet (X, Y, S) de sous-parties de V, disjointes deux à deux, on note $\langle X|S|Y\rangle_G$ la propriété "X et Y sont séparés par S dans le graphe G".

La séparation permet d'introduire une relation ternaire sur les sous-ensembles de noeuds d'un graphe.

Définition 19 (*I-MAP, D-MAP, P-MAP, Graphe Isomorphisme*)

Soit $G = (V, E)$ un graphe et $M \subset T(V)$ un modèle d'indépendance,

- *G est une D-map de M* *ssi* $(X \perp Z|Y) \Rightarrow \langle X|Y|Z\rangle_G$
- *G est une I-map de M* *ssi* $(X \perp Z|Y) \Leftarrow \langle X|Y|Z\rangle_G$
- *G est une P-map de M* *ssi* $(X \perp Z|Y) \Leftrightarrow \langle X|Y|Z\rangle_G$

Si un graphe G est une D-map d'un modèle M, toute connexion de sous-ensembles de noeuds indique une dépendance dans M. Si le graphe est une I-map, toute séparation est alors l'indication d'une indépendance dans le modèle. Une P-map est à la fois une D-map et une I-map.

B.3.1 Séparation dans les graphes non orientés

Définition 20 (*Séparation non orientée : u-séparation*)

Soit $G = (V, E)$ un graphe non orienté ; pour tout triplet (X, Y, Z) de sous-ensembles dijoints de G, X est u-séparé de Y par Z dans G (noté $\langle X|Z|Y\rangle_G$) **si et seulement si** *toute chaîne d'un noeud de X vers un noeud de Y passe par un noeud de Z. Un chemin qui contient un noeud de Z est appelé* ***bloqué*** *(par Z), sinon, il est appelé* ***actif****.*

$$\langle X|Z|Y\rangle_G \Leftrightarrow \left[\forall (x_i)_{i\in\{1...p\}\subset N} \text{ chaine de } G, x_1 \in X x_p \in Y \Rightarrow \exists i \in \{1...p\}, x_i \in Z\right]$$

B.3.2 Séparation dans les graphes orientés : d-séparation

La séparation dans les graphes orientés (d-séparation) est plus complexe que dans les graphes non-orientés (u-séparation). Cette dernière prend en compte la direction des arcs entre les variables.

Définition 21 (*Puits et V-structures*)

Soit une chaîne $C = (x_i)_{i\in I}$ dans un graphe orienté $\overrightarrow{G}$. On dira que x_i est un puits *de la chaîne C s'il est du type : $x_{i-1} \rightarrow x_i \leftarrow x_{i+1}$; c'est-à-dire s'il est un sommet à arcs convergents dans la chaîne.*

On appelle V-structure *de sommet puits C, tout sous-graphe de la forme $x_{i-1} \rightarrow x_i \leftarrow x_{i+1}$ où x_{i-1} et x_{i+1} ne sont pas connectés.*

Définition 22 (*Chaîne active, bloquée*)

Soit une chaîne $C = (x_i)_{i\in I}$ dans $\overrightarrow{G}$ et Z un sous-ensemble de noeuds de $\overrightarrow{G}$. C est une chaîne active par rapport à Z si les deux conditions suivantes sont réunies :

- *Tout puits de C a l'un de ses descendants dans $\overrightarrow{G}$ ou lui même qui appartient à Z.*
- *Aucun élément de C qui n'y est pas un puits n'appartient à Z.*

Une chaîne non active par rapport à Z est dite bloquée par Z.

Définition 23 (*d-séparation*)

Soit $\overrightarrow{G} = (V, \overrightarrow{E})$ un graphe orienté sans circuits (DAG) ; pour tout triplet

(X, Y, Z) *de sous-ensembles disjoints de* $\vec{G}$, X *est u-séparé de* Y *par* Z *dans* $\vec{G}$ *(noté* $\left\langle X|\vec{Z}|Y \right\rangle_{\vec{G}}$*)* **si et seulement si** *toute chaîne* $(x_i)_{i\in\{1\dots p\}}$ *avec* $x_1 \in X$ *et* $x_p \in Y$ *est bloquée par* Z.

B.4 Propriétés de Markov et factorisation

Soit un graphe $G = (V, E)$ et une probabilité $\mathcal{P}$ sur l'ensemble des variables V. La question qui se pose ici est de pouvoir exploiter les propriétés graphiques (séparation) pour représenter les lois d'une probabilité (indépendance conditionnelle). *Comment alors trouver le rapport entre séparation et indépendance conditionnelle ?* Ce rapport permettra de mesurer la précision des informations que le graphe G apporte sur $\mathcal{P}$.

Cela se traduit par une série de propriétés dites de Markov qui sont plus ou moins fortes : plus le graphe vérifie de propriétés fortes, mieux il représente la loi de probabilité $\mathcal{P}$.

B.4.1 Cas des graphes non orientés

Définition 24 (*Propriétés de Markov*)

Le graphe non orienté G *et la loi de probabilité* $\mathcal{P}$ *peuvent vérifier :*

(P) ***la propriété de Markov par paire***
si et seulement si $\forall x, y \in V, x$ *et* y *non adjacents dans* G,

$$x \perp y | (V \setminus \{x, y\})$$

(L) ***la propriété de Markov locale***
si et seulement si $\forall x \in V$,

$$x \perp (V \setminus \bar{\vartheta_x}) | \vartheta_x$$

où ϑ_x *est le voisinage du noeud* x *et* $\bar{\vartheta_x}$ *est la fermeture de voisinage de* x, *c'est-à-dire* $\bar{\vartheta_x} = \vartheta_x \cup \{x\}$ *(voir Annexe A)*

(G) ***la propriété de Markov globale***
si et seulement si $\forall X, Y, Z \subset V$ *disjoints*,

$$\langle X|Z|Y \rangle_G \Rightarrow X \perp Y | Z$$

Définition 25 (*Réseau Markovien*)

Un graphe vériafiant (G) pour $\mathcal{P}$ est un réseau Markovien de $\mathcal{P}$.

Théorème 3

$$(G) \Rightarrow (L) \Rightarrow (P)$$

La propriété de Markov (P) ne demande que de tester chaque paire de variables non adjacentes alors que tester (G) demande un calcul sur un grand nombre de triplets de sous-ensembles de V. Alors une propriété du type (P)⇒ (L) ⇒ (G) serai plus interessante. Malheureusement, cela n'est pas toujours vrai.

Théorème 4

Si la loi de probabilité $\mathcal{P}$ vérifie la propriété suivante :

$$\forall X, Y, Z, W \ sous-ensembles \ disjoints \ de \ V,$$

$$Si \ X \perp Y | Z \cup W \ et \ X \perp Z | Y \cup W \ alors \ X \perp Y \cup Z | W$$

Alors, l'équivalence (G) ⇒ (L) ⇒ (P) est vérifiée.

Définition 26 (*Factorisation*)

Soit une loi de probabilité joint $\mathcal{P}(V)$ et un graphe non orienté $G = (V, E)$, $\mathcal{P}$ est dite ***factorisable*** *ou* ***décomposable*** *selon G* **si et seulement si** *pour tout sous-graphe complet S de G, il existe une fonction $\Psi_S(V)$ ne dépendant que des noeuds de S telle que :*

$$\mathcal{P}(V) = \prod_{S \ sous-graphe \ complet \ de \ G} \Psi_S(V)$$

Ψ_S est appelée un potentiel.

Propriété 1

Soit $\mathcal{C}$ l'ensemble des cliques de G. $\mathcal{P}$ se factorize selon G si et seulement si

pour toute clique C de $\mathcal{C}$, il existe un potentiel Ψ_C tel que :

$$\mathcal{P}(V) = \prod_{C \in \mathcal{C}} \Psi_C(V)$$

Comme toutes les probabilités ne se factorisent pas ainsi, on note :
$\mathcal{M}_F(G)$ l'ensemble des probabilités pouvant se factoriser ainsi selon G,
Pour une probabilité $\mathcal{P}$, (F) la propriété "$\mathcal{P} \in$" $\mathcal{M}_F(G)$.

Théorème 5

Une probabilité $\mathcal{P}$ pouvant se factoriser selon G vérifie alors la propriété globale de Markov (G), donc

$$(F) \Rightarrow (G) \Rightarrow (L) \Rightarrow (P)$$

De plus, si $\mathcal{P}$ est positive,

$$(F) \Leftrightarrow (G) \Leftrightarrow (L) \Leftrightarrow (P)$$

Alors, si un graphe non orienté $G = (V, E)$ est un graphe d'indépendance conditionnelle représentant une distribution de probabilité $\mathcal{P}$ (strictement positive) si et seulement si $\mathcal{P}$ est factorisable selon G.

B.4.2 Cas des graphes orientés

Définition 27 (*Propriété orientée de Markov*)

Le graphe orienté $\vec{G}$ et la loi de probabilité $\mathcal{P}$ vérifiant :

(OP) ***Propriété orientée de Markov par paire***
si et seulement si $\forall x, y \in V, x$ et y non adjacents dans $\vec{G}$ et $y \in nd(x)$,

$$x \perp y | (nd(x) \setminus \{y\})$$

(OL) ***Propriété orientée de Markov locale***
si et seulement si $\forall x \in V$,

$$\{x\} \perp nd(x) | \Pi_x$$

(OG) ***Propriété orientée de Markov globale***
si et seulement si $\forall X, Y, Z \subset V$ *disjoints*,

$$\left\langle X|\vec{Z}|Y \right\rangle_{\vec{G}} \Rightarrow X \perp Y|Z$$

avec $nd(x)$ *signifie non descendant de* x *et* Π_x *représente l'ensemble des parents de* x.

Tout comme dans le cas non orienté, la propriété de Markov globale (OG) représente la plus forte propriété que peut vérifier un graphe orienté pour y lire directement une indépendance conditionnelle.

Théorème 6

$$(OG) \Leftrightarrow (OL) \Rightarrow (OP)$$

Si de plus $\mathcal{P}$ *est positive, alors*

$$(OG) \Leftrightarrow (OL) \Leftrightarrow (OP)$$

Définition 28 (*Factorisation récursive*)

Soit une loi de probablité jointe $\mathcal{P}(V)$ *et un graphe orienté* $\vec{G} = (V, E)$. *On dit que* $\mathcal{P}$ *possède une factorisation récursive selon* $\vec{G}$ **si et seulement si** *pour tout noeud* X *de* G, *il existe une fonction* $k_X(X, \Pi_X)$ *telle que :*

$$\mathcal{P}(V) = \prod_{X \in V} k_X(X, \Pi_X)$$

Les fonctions k_X *sont les lois de probabilité conditionnelle de* X *sachant ses parants* Π_X *:*

$$\mathcal{P}(V) = \prod_{X \in V} \mathcal{P}(X, \Pi_X)$$

Comme toutes les lois de probabilité ne se factorisent pas ainsi, soit (OF) la propriété "$\mathcal{P}$ se factorise récursivement selon $\vec{G}$". La relation entre factorisation récursive et propriété orientée de Markov est la suivante :

Théorème 7

$$(OF) \Leftrightarrow (OG)$$

Alors, si un graphe orienté $\vec{G}= (V, \vec{E})$ est un graphe d'indépendance conditionnelle représentant une distribution de probabilité $\mathcal{P}$ si et seulement si $\mathcal{P}$ est factorisable selon $\vec{G}$.

'Étant donnée une loi de probablité $\mathcal{P}$, il n'y a pas forcément un unique graphe $\vec{G}$ permettant la factorisation récursive de $\mathcal{P}$, on appelle ***classe d'équivalence de Markov*** l'ensemble des graphes permettant une factorisation récursive de $\mathcal{P}$, c'est-à-dire l'ensemble des graphes représentant le même modèle d'indépendance.

Définition 29 (*Équivalence au sens de Markov*)

Deux DAG sont équivalents au sens de Markov (noté $\equiv$) s'ils représentent les mêmes relations d'indépendance conditionnelle.

Théorème 8 (*[VP91]*)

Deux DAG sont équivalents si et seulement si *ils ont le même* squelette *et les mêmes* V-structures.

Modèles Graphiques Probabilistes pour l'Estimation de Densité en grande dimension : applications du principe Perturb & Combine pour les mélanges d'arbres

Résumé :
Dans les applications actuelles, le nombre de variables continue d'augmenter, ce qui rend difficile l'estimation de densité. En effet, le nombre de paramètres nécessaire pour l'estimation croit exponentiellement par rapport à la dimension du problème. Les modèles graphiques probabilistes fournissent une aide non négligeable pour lutter contre ce problème en fournissant une factorisation de la loi jointe mais souffrent d'un problème de passage à l'échelle.
Le problème de grande dimension s'accentue du fait que le nombre d'observations avec lequel on effectue l'estimation de densité n'augmente pas dans les mêmes proportions, et reste même extrêmement faible dans certains domaines d'applications. La factorisation de la loi jointe s'avère non suffisante pour effectuer une estimation de densité de qualité lorsqu'il y a très peu de données. Le principe du *Perturb & Combine*, initialement appliqué en classification, permet de lutter contre ce genre de problèmes.
Dans le cadre de cette thèse, nous proposons un algorithme générique d'estimation de densité en appliquant le principe du Perturb et Combine à une famille de modèles graphiques probabilistes "simples" , les structures arborescentes "manipulables" avec une complexité au pire quadratique.
Plusieurs variantes de cet algorithme sont proposées en exploitant à deux niveaux le principe de perturbation: perturbation de la génération des modèles simples et perturbation des données d'apprentissage.
Les expérimentations effectuées lors de ce travail montrent que nos premières approches sont concluantes en ce qui concerne la qualité d'approximation, pour une complexité algorithmique quadratique encore insuffisante en grande dimension.
Notre seconde contribution concerne donc une nouvelle application du principe de perturbation, permettant d'arriver à une complexité algorithmique proche du quasi-linéaire pour une même qualité d'approximation.

Mots-clés : Estimation de densité, modèles graphiques probabilistes, Perturb & Combine, données éparses, espaces de grande dimension.

Probabilistic graphical models for density estimation in high dimensional spaces: application of the Perturb & Combine principle with tree mixtures

Abstract :
The dimensionality of current applications increases which makes the density estimation a difficult task. Indeed, the needed number of parameters to make estimation grows exponentially with respect to the dimension of the problem. Probabilistic graphical models can be used to solve this problem by providing a factorization of the joint distribution, but they suffer from a problem of scalability. The problem of high dimensional spaces is accentuated by the number of observations used to perform density estimation witch is not increased in the same proportions, and even remains extremely law in some applications.
Factorization of the joint distribution is not sufficient to perform good density estimation with sparse data. The Perturb and Combine framework, first explored in classification, provide solutions for such problems.
In this work, we explore and propose a generic algorithm for density estimation by applying the Perturb and Combine principle to a reduced family of simple probabilistic graphical models. These tree structures we proposed to use can be "manipulated" with at worst a quadratic complexity. Several variants of this algorithm are proposed by exploiting the Perturb and Combine principle according to two levels: perturbation of the tree generating procedure and perturbation of the learning dataset.
Our initial approaches are conclusive regarding the quality of approximation, with a quadratic computational complexity, still insufficient in high dimensional spaces. Our second contribution concerns therefore a new application of the Perturb and Combine principle, which allows attending almost quasi-linear computational complexity, for the same quality of approximation.

Keywords: Density estimation, probabilistic graphical models, Perturb &Combine, high dimensional sparse data.

Discipline : Informatique